NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

An Automated Tool To Facilitate Code Translation for Software Fault Tree Analysis

by

Robert R. Ordonio

September 1993

Thesis Advisor: Dr.Timothy J. Shimeall

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1993	Master's Thesis, July 1991 - September 1993

4. TITLE AND SUBTITLE
An Automated Tool to Facilitate Code Translation for Software Fault Tree Analysis (U)

5. FUNDING NUMBERS

6. AUTHOR(S)
Ordonio, Robert Romero

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Unclassified/Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A safe system is defined as a system that prevents unsafe states from producing safety failures, where an unsafe state is defined as a state that may lead to safety failure unless some specific action is taken to avert it. The problem that this thesis addresses is how to find places in Ada programs where faults are likely to occur during program execution.

The approach is to build an automated translation tool that translates Ada programs into a software fault tree. [Lev 83] The tool works as follows: 1). The Ada parser and lexical analyzer calls the Automated Code Translation Tool (ACTT) upon recognition of an Ada statement; 2). The ACTT produces a template representing the statement; 3). The templates are linked together as a software fault tree.

The result is a program that takes Ada source code as input and produces a software fault tree as output.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Ada, Fault Tree Analysis, Fault Tree Editors, Lexical Analyzers, Parsers, Safety, Software Fault Tree Analysis, Software Safety, System Safety.	212
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

Approved for public release; distribution is unlimited

An Automated Tool To Facilitate Code Translation for Software Fault Tree Analysis

by

Robert Romero Ordonio
Captain, United States Army
B.S., McIntire School of Commerce
University of Virginia, 1984

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1993

Author: _____
Robert Romero Ordonio

Approved By: _____
Dr. Timothy J. Shimeall, Thesis Advisor

_____
Dr. David A. Erickson, Second Reader

_____
Dr. Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

A safe system is defined as a system that prevents unsafe states from producing safety failures, where an unsafe state is defined as a state that may lead to safety failure unless some specific action is taken to avert it. The problem that this thesis addresses is how to find places in Ada programs where faults are likely to occur during program execution.

The approach is to build an automated translation tool that translates Ada programs into a software fault tree. [Lev 83] The tool works as follows: 1). The Ada parser and lexical analyzer calls the Automated Code Translation Tool (ACTT) upon recognition of an Ada statement; 2). The ACTT produces a template representing the statement; 3). The templates are linked together as a software fault tree.

The result is a program that takes Ada source code as input and produces a software fault tree as output.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank a number of people, for without their assistance, guidance, and especially patience, this thesis would not have been completed. First, I would like to thank Professor Shimeall for the endless support he gave me throughout the entire process. His willingness to provide assistance in solving problems and his constant availability throughout were key contributing factors to this thesis completion. Additionally, I would like to thank Professor Erickson for his contributions in making this thesis more understandable. Thanks are also due to Dr. Friedman and Mr. Lombardo for their previous works on automated code translation and fault tree editors, respectively.

Last, but most definitely not least, I would like to especially thank my wife, Gemma. Her loving support, patience, understanding, and encouragement during our stay in Monterey provided me the inspiration to finish this project.

I. INTRODUCTION

Since the invention of computers, from main frames to the micro systems, the appearance and usage of these devices in today's society has grown at a tremendous pace. Computers today are used in all aspects of everyday life. The range of devices that use computers include the complicated systems of the space shuttle and different weapon systems to the less intricate systems of calculators and small appliances. With the introduction of computer systems, there exists the requirements to ensure that these systems will perform as intended without catastrophic side effects. In essence, the goal is to ensure that the computer systems operate safely. A system is defined by *The American Heritage Dictionary* as a group of interacting, interrelated, or interdependent elements forming a complex whole.

A. SAFETY

Safety has been defined as a state that is free from conditions that can cause death, injury, occupational illness, or damage to or loss of equipment or property. [Lev 86] Leveson, along with others, have stated that this definition is unrealistic because by this definition any system that presents any element of risk would be labeled unsafe. [Lev 86] Risk is defined as a function of the probability of the hazardous state occurring, the probability of the hazard leading to a mishap, and the perceived severity of the worst potential mishap that could result from the hazard. [Lev 86]

Safety can be used as a relative term depending on the situation. An item may not be safe under all circumstances and conditions, but compared to the alternatives to that item, it may be the safest choice available. For example safety razors and safety matches by their very nature poses some risk; however, they are safer then their alternatives and they provide benefits to the user at a more acceptable level of risk then their alternatives. [Lev 86]

In the development of a system, trying to eliminate all unsafe or hazardous conditions may be impossible. Instead of removing the risk, a displacement of the risk to another area

is frequently the result. In these situations one must use a risk/benefit analysis to determine what is the best solution. For example, nitrate, which is used in foods, have been linked to cancer. However, nitrate is also used to prevent botulism. "Benefits and risks often have trade-offs, such as trading off the benefits of improved medical diagnosis capabilities against the risks of exposure to diagnostic X-rays." [Lev 86] The trade-offs between the benefits and the risks are dependent on the particular situation and are usually defined by political, moral and economical decisions of the users.

Safety depends on the situation in which it is measured. Leveson stated that nothing is completely safe under all conditions. Items that may be relatively safe under most conditions could become hazardous if the right conditions exist. An example of this is drinking too much water, which could in turn, lead to kidney failure. [Lev 86]

To understand safety, one must distinguish safety from reliability. Reliability pertains to the probability that a system will perform its intended function for a specified period of time under a set of specified environmental conditions, thus making a system failure free. [Sal 76] Safety is concerned with the possible errors that may lead to a hazard and reliability looks at all possible errors. Since not all errors cause hazards, safety requirements only look at the subset of the errors that cause the undesired events. [Lev 86]

According to Brown, safety is not reliability, availability or any other attribute or combination of attributes. In some situations, safety and reliability may actually counter each other. If the design is unsafe to start with, no degree of reliability is given to make it safe. Along with this, if a system fails to a safe state, it is not reliable but it is safe. [Bro 88]

Leveson states that safety and reliability can complement each other or can have no direct relationship with each other. A hydraulic system of an aircraft is more safe when it is more reliable. On the other hand, measuring reliability for munitions pertains to detonation at the "desired time and place, while safety is related to inadvertent functioning". For reliability and safety in this example, there is no direct relationship. [Lev 86]

In addition, another aspect of reliability, availability, does not equate to safe. Availability is the probability that a system or component is operational at a specific time, available when needed. [Sal 76] "A system may be safe but not available and may also be available but unsafe (e.g., operating incorrectly)." [Lev 86]

B. SYSTEM SAFETY

System safety pertains to the safety of the system, both hardware and software. A safe system is defined as a system that prevents unsafe states from producing safety failures, where an unsafe state is defined as a state that may lead to safety failure unless some specific action is taken to avert it. [Lev 86] In the past hardware-oriented system engineers have treated the computer as a black box because of the "newness" of software engineering and the "significant difference between software and hardware". [Lev 86] Software engineers on the other hand "treated the computer as merely a stimulus response system". [Lev 86] According to Leveson "the greatest cause of problems experienced when computers are used to control complex processes may be a lack of system-level methods and view points". [Lev 86] In order to better resolve problems with systems, these old views must be changed to an outlook that views the system as a whole. Leveson stated, "...that software itself is not 'unsafe'. Only the hardware that it controls can do damage." [Lev 86]

System safety is a subdiscipline of systems engineering that applies scientific management and engineering principles to ensure adequate safety throughout the system life cycle, within the constraints of operational effectiveness, schedule, cost and performance. [Lev 86] The primary goal of system safety is to develop an acceptable safety level into the product before production or operations occur. This goal can be achieved in two steps. First, hazards need to be identified by applying scientific and engineering principles. Along with this, hazards need to be controlled through analysis, design and management procedures. The second step is to eliminate the hazards that post an

unacceptable risk from the system design. [Lev 86] Even though this is simply stated, the process is very complex.

C. SOFTWARE SAFETY

In the previous section the overall safety of the system was considered. This section takes a look at specifically the software issue because hardware reliability has improved impressively in the last few years and they are no longer a major cause of system failures. According to Cha, software has become the major "bottleneck" in further improving the system. He also stated that perfectly safe software is impossible and un-necessary and that a more realistic goal is to develop software that is free from hazardous behavior. [Cha 91]

Brown defines software safety as the optimization of system safety in the design, development, use and maintenance of software systems and their integration with safety critical hardware systems in an operational environment. [Bro 88] The intent of software safety is to ensure the software will function in a system context without creating an unacceptable level of risk.

Software safety, however, is difficult to achieve. All methodologies in developing software use humans and humans are fallible. [Lev 83] According to Brown, virtually all errors that occur in software may be attributed to humans. These errors can be grouped into four categories: design, coding and testing, operator, and maintenance. [Bro 88] Additionally, "software...has a large number of discrete states without the repetitive structure found in computer circuitry. Although mathematical logic can be used to deal with functions that are not continuous, the large number of states and lack of regularity in the software results in extremely complex mathematical expressions." [Lev 86]

Along with this, software safety is difficult because it is hard to develop realistic test conditions. Even with the use of simulations, it is difficult to guarantee that the simulation is accurate. "Software faults may not be detectable except under just the right combination of circumstances, and it is difficult, if not impossible, to consider and account for all

environmental factors and all conditions under which the system may be operating." [Lev 86]

Problems with the requirement specifications can also pose a problem to software safety. In a study of computer mishaps by Erickson and Griggs, it was determined that inadequate design foresight and specification errors are the greatest cause of software safety problems. [Eri 81] [Gri 81] In performing tests on software, the test can only consider the specified requirements and can not look at what was intended by the designer.

D. STATEMENT OF PROBLEM

Although there exists potential problems and several unknowns in the use of computers in systems that control critical devices, the advantages that computers provide including increased versatility and power, improved performance, greater efficiency, and decreased cost, lend merit to their use in these systems. Computer systems consist of two major components, hardware and software. Even though new discoveries in hardware are occurring constantly, there exist techniques that allow engineers to better handle the problems that arise in hardware. Engineers working with hardware are better able to anticipate the possible problems with hardware because the development of hardware follows long-established procedures. Software, on the other hand, is more complex than hardware because of the large number of states present in software. Additionally, software is normally developed for a particular application only using tests in simulations and controlled environments as a basis for reliability. [Lev 86]

Analyzing software with a purely manual technique is expensive because the step to traverse through the code requires an extra-ordinary amount of time. In addition, performing any function manually may introduce errors because humans are fallible. Automating the entire process poses other problems. One problem with automating the entire process is the loss of human insight required to see the intricacies of the system. Additionally, the experience of the analyst is lost when the process is fully automated. To

achieve a proper balance of manual and automated techniques in software safety analysis, both techniques are required.

This research pertains to software safety analysis, specifically increasing automation of the process. Introducing automated methods into software analysis may provide the analyst the opportunity to use his time more efficiently. The removal of the detailed work of code translation allows the analyst to focus on the semantics of the analysis, not the syntax of code. The primary purpose of this research is to assist in automating the process of translating code into usable structures for software fault tree analysis.

Software fault tree analysis is a method of software safety analysis that looks at the logic of the software. In this process, an undesired event caused by the software is identified and is labeled as the top node. The process works by looking at the top node and identifying the preconditions that are required for the top node to occur, given the design of the software. This process continues for each node at each level until the current node can not be further decomposed.

This thesis introduces a prototype tool that will translate Ada statements into pre-defined template structures that are used in software fault tree analysis. This tool performs a static analysis of Ada code using Ayacc, an Ada parser generator; Aflex, an Ada lexical analyzer generator; FTE, a graphical fault tree editor; and self-generated code. After inputting the Ada code into the tool, the tool will basically parse the given Ada input and upon recognition of various Ada structures, build pre-defined templates. Upon completion of the parsing and template generation, an FTE-compatible file will be generated.

One question addressed by this research looks at the ability to automate the software fault tree analysis. Introducing automation to most processes reduces the time required to complete the process since machine is inherently faster than man. If automating software fault tree analysis provides more benefits compared to costs, then it may be of value to implement some automation to the process.

Another question addressed pertains to the cost of automating software fault tree analysis. A purely manual technique is too expensive, but a purely automated process also

has problems. If a tool can be developed that strikes a balance between manual and automated techniques, the cost of automating software fault tree analysis could be reduced.

E. SUMMARY OF CHAPTERS

Chapter II provides background information that is related to this thesis. Areas covered in this chapter are fault tree analysis, software fault tree analysis, and previous works in this area including fault tree editors, the use of templates to represent statements, and the automation of code translation for software fault tree analysis. Chapter III covers the actual tool developed. Information on Ayacc and Aflex are also provided. Additionally, an extended example is reviewed in this chapter. Chapter IV provides the conclusions including the recommendations and provides a summary of possible future work. Appendix A contains the Ayacc and Aflex specification file input. Appendix B is a listing of the source code excluding the generated code by Ayacc and Aflex. Appendix C provides the templates for individual Ada structures. Appendix D contains the output of the Automated Code Translation Tool when the example procedure FAKE_OVEN_CONTROL was inputted into the tool.

II. BACKGROUND INFORMATION

A. FAULT TREE ANALYSIS

Fault tree analysis (FTA) is a method of deductive safety analysis where a fault tree depicts the logical interrelations of basic events that lead to a particular event of interest. [Lev 91] FTA originated initially in 1961 at Bell Laboratories to aid in the analysis of the Minuteman missile. Boeing further developed FTA in the mid 1960's. [Sal 76] A fault tree consists of a fault event, a failure situation resulting from the logical interaction of primary failures or failures of interest; a branch, the connection between two fault events or fault tree gates; and a fault tree gate, the boolean logic symbol that relates the input to its output event. [Fus 73] The fundamental concept of FTA is to decompose a physical system into a fault tree, also known as a logic diagram, where certain specified causes lead to one specified event of interest. This specified event of interest is also known as the "Top" event. [Sal 76]

To develop a fault tree, the initial step is to define the top event that will be analyzed. After the top event is defined, the next step is to locate the event(s) that by themselves, or in combination with others lead to the top event. This analysis looks for a plausible sequence of events that can lead to the event of interest in the context of the system's environment. [Mcg 89] If one or more event(s) is/are required to produce the event of interest, an And Gate is used to connect the event(s) to the event of interest. If one or more event(s) can lead to the event of interest, the event(s) is/are connected to the event of interest with an Or Gate. This process will continue until all events in the tree are defined or are not decomposable. [Sal 76] The results of the fault tree either depict the method of how the top event could occur or show that the top event can not occur. [Mcg 89] For a more detailed discussion on fault trees see the papers by Salem and Fussell. [Sal 76] [Fus 73]

B. SOFTWARE FAULT TREE ANALYSIS

McIntee coined a phrase "Soft Tree" in 1983 for a process which looked at software analysis using FTA techniques. This technique required a schematic of the system and a flow diagram of the software. His work basically started by identifying a top event and worked backward through the schematic and flow diagram. The result of this process provided a detailed path for how the process worked from the top to the bottom and through the hardware and software. [Mci 83]

Taylor in 1982 [Tay 82] and Leveson and Harvey in 1983 [Lev 83] provided initial information on software fault tree analysis (SFTA). SFTA is a technique to analyze the safety of a software design that is similar to Soft Tree which uses methods similar to FTA. SFTA looks at the logic contained in the software. The primary purpose of SFTA is to ensure that the logic of the software does not produce system safety failures. Additionally, SFTA determines environmental conditions that could cause the software to create a safety failure. [Lev 83]

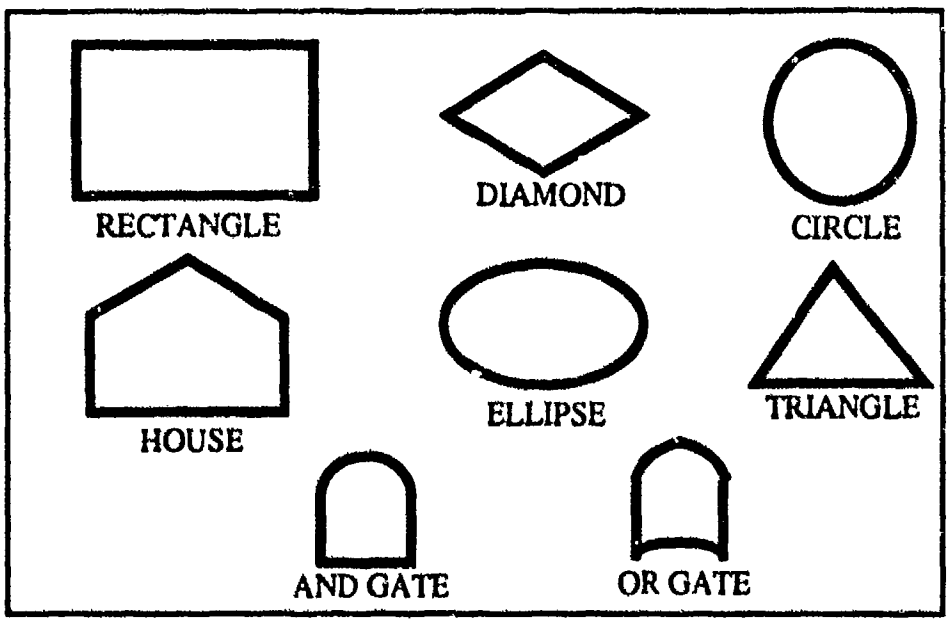


Figure 1 Common Software Fault Tree Symbols

SFTA, like FTA, is designed to work from the identified hazard backwards to the initial command to execute the system. In addition, SFTA uses a subset of the symbols used

for FTA. (See Figure 1) The "rectangle" represents an event that will be further analyzed. The "diamond" is used for nonprimal events, which are not developed further for lack of information or insufficient consequences. The "circle" indicates an elementary event or primary failure of a component where no further development is required. The "house" is used for events that normally occur in the system. It represents the continued operation of the component. The "ellipse" is used to indicate a state of the system that permits a fault sequence to occur. This system state may be normal or a result of failures. The "triangle" represents another sub-tree for the node which is not depicted on the current tree. The "and gate" serves to indicate that all input events are required in order to cause the output event. The "or gate" indicates that any of the input events may produce the gated event. [Rol 86]

This common use of symbols allows for the interfacing of SFTA and FTA, thus providing a means for systems analysis of the hardware and software. The process of SFTA is similar to FTA, the difference is that SFTA pertains to software events and that SFTA does not employ the probabilistic approach of FTA, but instead deals with logic. "The basic procedure is to assume that the software has performed in a manner, which has been determined by the hazard analysis will lead to a catastrophe, and then to work backward to determine the set of possible causes." [Lev 91] After determining the catastrophe that occurred, the process works backward to the next level to determine the necessary preconditions for the catastrophic event to occur. Each child of each node will encounter the same process until the child can not be analyzed for some reason. The result of this process is a tree diagraming how an undesired event could occur or proof that an event could not occur. [Lev 91]

SFTA provides many advantages to software analysis. The advantages include the following:

- Provides the focus needed to give priority to catastrophic events and to determine the environmental conditions under which a correct state becomes unsafe.
- Provides a convenient structure to store the information gathered during the analysis which can be used later for redesign.

- The technique is familiar to hardware designers.
- Provides a single structure for specifying software, hardware, human actions and interfaces with the system.
- Allows the examination of the effects of underlying machine failures or environmental changes versus verification techniques which assume system operates correctly. [Lev 83]

C. PREVIOUS WORK

1. FAULT TREE EDITORS

Editors to graphically display fault trees have been developed to facilitate the drawing of fault trees by the analyst. These tools allow the user to interactively modify the graphic representation of a fault tree using an automated graphic editor. The use of automated editors reduces the time required to draw the trees by removing the manual tasks necessary. Additionally, some of the errors that may have been caused by the manual drawing can be eliminated with an automated graphic editor. Two tools reviewed during the research of this thesis are the Software Fault Tree Analysis Tool [Rol 86], developed at the University of California, Irvine and Fault Tree Editor (FTE), developed at the Naval Postgraduate School. This thesis used FTE because it was readily available and operational at the time this thesis was initiated.

The Fault Tree Editor is an interactive fault tree design tool. FTE was written by Charles P. Lombardo, a computer systems programmer at the Computer Science Department at the Naval Postgraduate School in Monterey California. FTE was developed using the C programming language and XView, an OPEN LOOK toolkit for the X11 Windowing system.

Once FTE is running, the user can load a file to display a fault tree. The file can either be a file created by the tool or a file that meets the specifications of FTE. The specification of an FTE file consists of four lines for each node on the tree. The first line is a string of characters that represent the label of the node. This value is the label displayed on the node within the tree. The second line is another string of characters that represent the description of the fault. The third line is also a string of characters. This string of characters

is the file name of the code associated with the node. The fourth line contains seven integer fields that are separated by any white space except a new line. These values represent the starting and ending line numbers of the code associated with the node, the X and Y coordinates where the node is located in the FTE graphical drawing field, the type of node, the type of gate attaching the node to its children, and the number of children of the node. The file is arranged with the four lines of each node grouped together and each node ordered in a pre-order fashion according to its relationship to the root node. (See Figure 2)

```
-- Node Values
Label:          "Top"
Fault:          "Node caused fault"
File:           "Example.a"
Start Line:     1
End Line:       55
X Coordinate:   200
Y Coordinate:   300
Type Node:      1
Type Gate       0
No. Children:   0

-- The above node in FTE type specification

Top
Node caused fault
Example.a
55 200 300 1 0 0
```

Figure 2 Example Node Values

2. SOFTWARE FAULT TREE TEMPLATES

Taylor discussed the building of fault trees and software fault trees in his paper in 1982. Using techniques from FTA and SFTA discussed above, Taylor used finite state models of plant components in the development of the fault tree. Taylor's process used the

plant components, also known as "mini-templates", in building the fault trees. At each level, the event being analyzed would be matched to the mini-template to determine the events required to cause the undesired event. The process terminated when a spontaneous or normal event was encountered.

Leveson, Cha, and Shimeall wrote several papers on SFTA in 1987, 1989, and 1991. Their work provided a manual means to perform SFTA for Ada code. Their work used predefined templates as the failure semantic of the Ada programming language statements. These templates were based on Ada statement structures, showing that if a particular Ada structure caused a fault, where could that structure have caused the fault. By linking the templates according to the code, the software fault tree generated would provide a tree depicting where faults, if they occurred, could occur. [Lev 91] Modified versions of the templates are in Appendix C.

3. AUTOMATED SOFTWARE FAULT TREE ANALYSIS

Friedman automated SFTA for certain code structures of Pascal in 1993. His procedure consisted of four steps. Initially, a lexical analysis of the input would remove the blanks, tabs, and carriage returns and breakdown the code into distinguishable items called tokens. The second step involved the parsing of the tokens. This parsing step would generate a list structure that described the syntactic structure of the program. Upon completion of the parsing step, the list would be traversed creating the software fault tree using pre-defined templates and rules. This step would produce an ASCII file that would be compatible to a proprietary editor tool so that the tree could be drawn. The final step is the actual drawing of the tree using the generated ASCII code and the editor tool. Friedman's tool used a commercial software product called Tree-Master that allowed the analyst to edit, display, and print the tree. [Fri 93]

Friedman's research provides an introduction of automation to the software fault tree analysis methodology of software analysis. His tool, however, only looks at a limited number of code structures of Pascal, namely the assignment-statement, the if-statement,

and the while-statement. This research looks at the use of automation in the code translation of Ada code since Ada code is used in control systems more than Pascal and Ada code contains unique structures, such as exceptions and tasking.

III. AUTOMATED CODE TRANSLATION TOOL PROTOTYPE

In the past, the code translation for SFTA has normally been performed using manual techniques. The use of manual techniques is a timely proposition. In addition, the use of manual techniques can lead to the introduction of errors because humans are fallible. This thesis introduces a prototype for an automated code translation tool to minimize the amount of manual code translation needed for SFTA. This tool is intended to reduce the amount of time necessary and reduce the number of errors in translating code by limiting the amount of human involvement. This chapter will discuss the tool, detailing the items used in the development of the tool and the methodology used in building the tool.

The Automated Code Translation Tool, as its name suggests, is a tool that will translate Ada statements into template structures to be used in SFTA. The tool consists of basically four components. The first component is a lexical analyzer. The lexical analyzer will determine if the given input consists of valid tokens. The next component is a parser. This part of the tool will check the given input to ensure that valid Ada constructs are used. The third component is a template generator that will transform valid statements into templates representing possible events associated with the statement in a format suitable for SFTA. The final component of the tool is a file generator that will create a file that meets the specifications of an FTE file type. Each component will be discussed in the following sections. (See Figure 3)

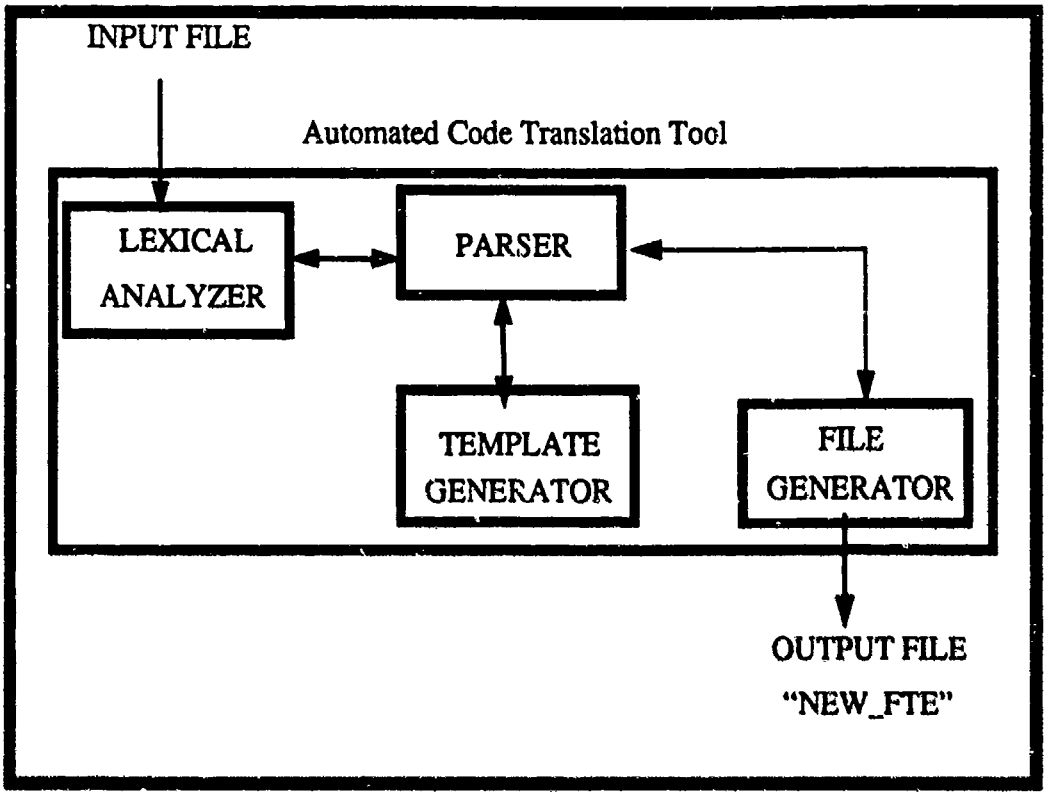


Figure 3 Automated Code Translation Tool Structure

A. THE LEXICAL ANALYZER AND PARSER

A lexical analyzer is a part of a system that breaks up the inputted code into meaningful units called tokens. Tokens can be names, constants, reserved words, or operators, among other things that are defined by the programming language. The lexical analyzer will also remove white space including blank spaces, empty lines, and carriage returns. The primary purpose of the lexical analyzer is to determine if the given input consists of valid tokens for the programming language. [Ill 90] The lexical analyzer for the Automated Code Translation Tool was built using an Ada based lexical analyzer generator called Aflex. For more information on Aflex, see the Aflex user manual. [Ngu 88]

The specification file for Aflex used to develop the lexical analyzer in the Automated Code Translation Tool was developed by Herman Fischer of Litton Data Systems in March 1984. The lexical input information in this file represents the grammar for ANSI Ada. No

modifications to the file were required in the development of the tool. The Aflex specification file for the Automated Code Translation Tool is located in Appendix A.

A parser is the programming module that performs parsing operations on a computer language. Parsing is the process of deciding whether a string of input symbols is a sentence of a given language, and if so, determining the syntactic structure of the string as defined by a grammar for the language. The primary purpose of the parser is to validate the string as a correct statement for a given language. [Ill 90] The parser for the Automated Code Translation Tool was built using an Ada based parser-generator called Ayacc. For more information on Ayacc, see the Ayacc user manual. [Tab 88]

The Ayacc specification file used to develop the parser in the Automated Code Translation Tool was a modified version of the David Taback and Deepak Tolani specification file. The Taback and Tolani version was an adaptation of the Herman Fischer file used in Yacc. The grammar for the file is organized in the same order as the ANSI Ada Reference Manual.

The modifications to the Taback and Tolani file for the tool consisted of added actions to be performed when particular code structures were recognized. These actions are related to functions and procedures located in separate packages and are components of the template generator which will be discussed in the next section. The packages are FAULT_TREE_GENERATOR and TRAVERSE_PKG and are located in Appendix B. The specification file for Ayacc used in the development of the parser for the Automated Code Translation Tool is located in Appendix A.

B. THE TEMPLATE GENERATOR

Basically the templates represent the idea "If this statement caused a fault, where could that fault have been introduced within the statement". These templates were developed using the previous work of Leveson, Cha, and Shimeall. [Lev 91] The templates were slightly modified to accommodate the parsing of the Ada grammar. The templates for the

implemented code structures are located in Appendix C. The symbols used in the templates are similar to the symbols used in software fault trees. (See Figure 1)

The template generator consists of functions and procedures that are called when various Ada structures are recognized. The template generator works in concert with the parser. As the parser identifies various structures, the template generator will be called to build the template for the particular structure. This process is exemplified with an if-statement example. (See Figure 4) The following is a description of how the if-statement is parsed using the Ada grammar and how the template is built with the tool.

```
IF temperature < 15.0 THEN
    rad_set:= rad_set + 5.0;
ELSIF temperature < 18.0 THEN
    rad_set:= rad_set + 1.0;
ELSE
    rad_set:= rad_set - 1.0;
END IF;
```

Figure 4 If-Statement Example

When the parsing of Ada source code begins, it initially starts at the starting non-terminal "compilation". Eventually it proceeds to the sequence of statements non-terminal and then to the statement non-terminal. Using the current token "if", the if-statement grammar rule is selected and the process to build the template for the if-statement begins. The if-statement grammar rule consists of seven grammar symbols. (See Figure 5) Before the if-statement template is built, the parser must identify and parse the individual grammar symbols. This discussion looks at only the parsing of the non-terminals because terminal symbols are recognized by lexical analysis. (When looking at the grammar rules, the capital words identify non-terminals in the rule.)

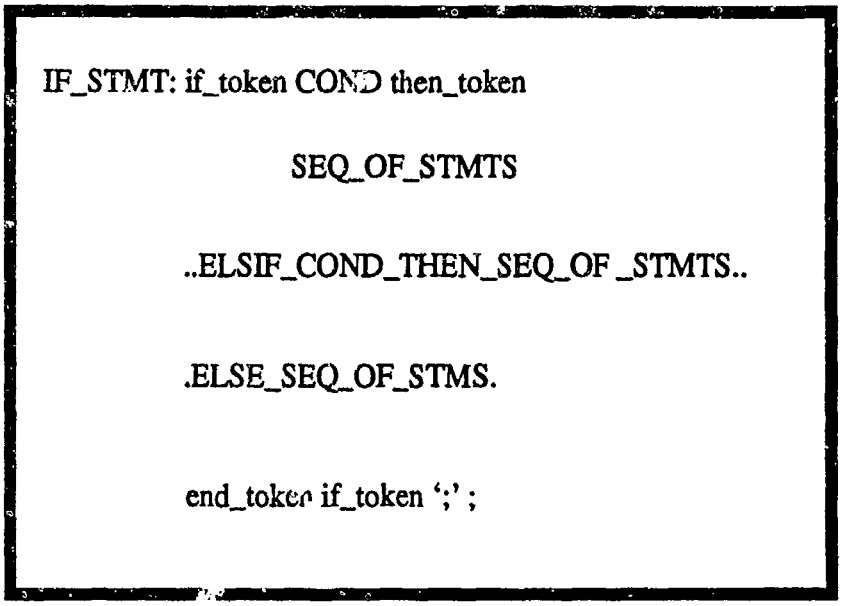


Figure 5 Production Rule for If-Statement

The first non-terminal of the if-statement rule is the condition non-terminal. The condition represents a boolean expression for the if-statement. In parsing the condition non-terminal, the expression non-terminal will be called. From the expression non-terminal, the relation non-terminal with one or more relations and zero or more logic operators, depending on the actual input, will be executed. At the relation level, a template containing possible events related to the actual condition will be built by the template generator and returned to the expression non-terminal level. This template will be passed backward to the condition non-terminal level and in turn passed back to the if-statement non-terminal level for use in the building of the if-statement template.

In the example, the condition is a simple relation, "temperature < 15.0". For this condition the template generator will build the simple relation template. (See Figure 6) This template would be passed back from the relation non-terminal back to the if-statement non-terminal through both the expression and condition non-terminals.

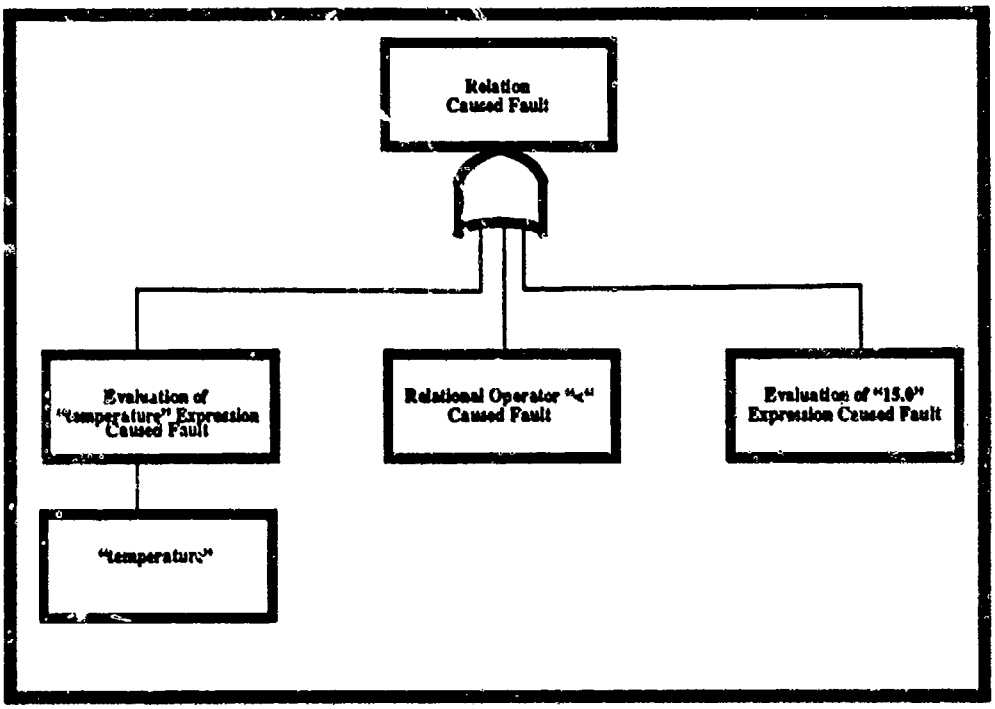


Figure 6 Relation Template for If Condition

Following the condition non-terminal, the sequence of statements non-terminal of the if-statement production rule would be parsed. This sequence of statements represents the action that would be performed if the condition evaluated to true. In the example, the parsing would again start at the sequence of statements non-terminal and recognize that the next non-terminal was a single statement. This statement would be identified as a simple statement, followed by recognition as an assignment-statement. At the assignment-statement production rule, the parser would parse the individual components of the rule. The rule consists of two non-terminals, name and expression. For the name non-terminal, the parser would recognize that "rad_set" is a simple name and at the simple name non-terminal the name template for "rad_set" would be generated by the template generator.

The next non-terminal of the assignment-statement is the relation, "rad_set + 5.0". This example is another simple relation. The relation template would be built for this expression by the template generator and passed back to the assignment-statement. (See Figure 7)

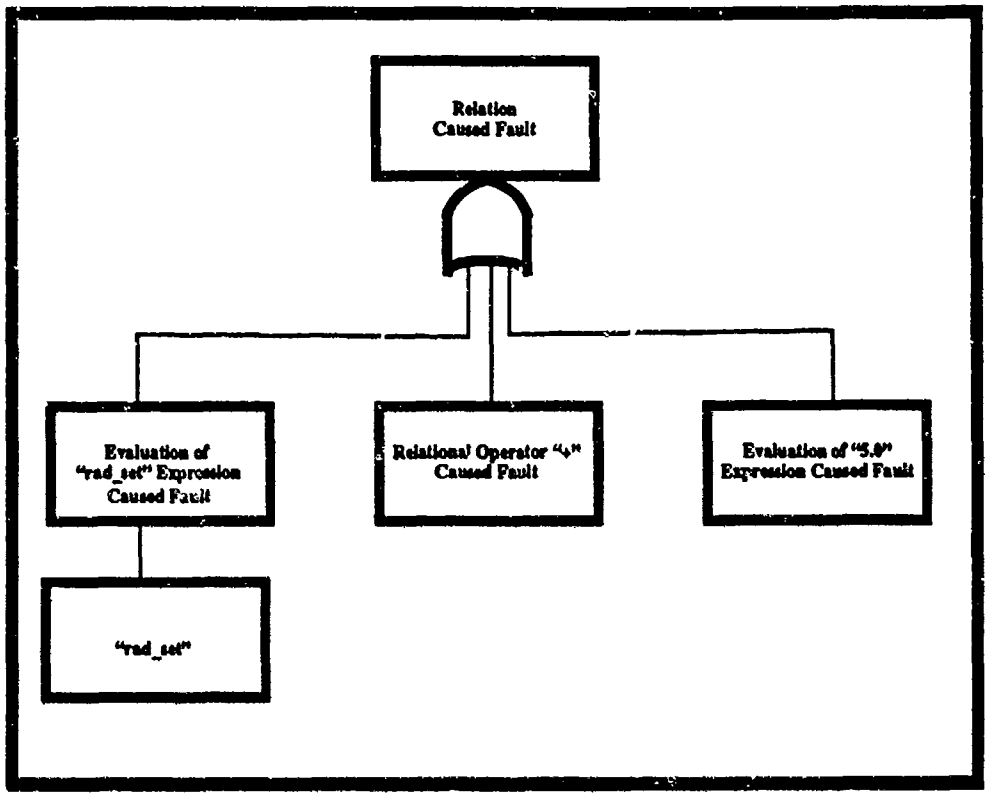


Figure 7 Relation Template for Assignment-Statement

When the parser returns to the assignment-statement non-terminal, the template generator will create the assignment-statement template with the simple name template and the relation template for the assignment-statement as components to the template. The two previously generated templates would be placed as children to the "Operand Evaluation Caused Fault" node. (See Figure 8) The assignment template will be incorporated into the template for the sequence of statements to represent the sequence of statements for the if condition. (See Figure 9) Since there is only one statement in the sequence of statements, the nodes pertaining to the previous statements will not be necessary. The sequence of statements template would be returned to the if-statement non-terminal level to be used in the creation of the if-statement template.

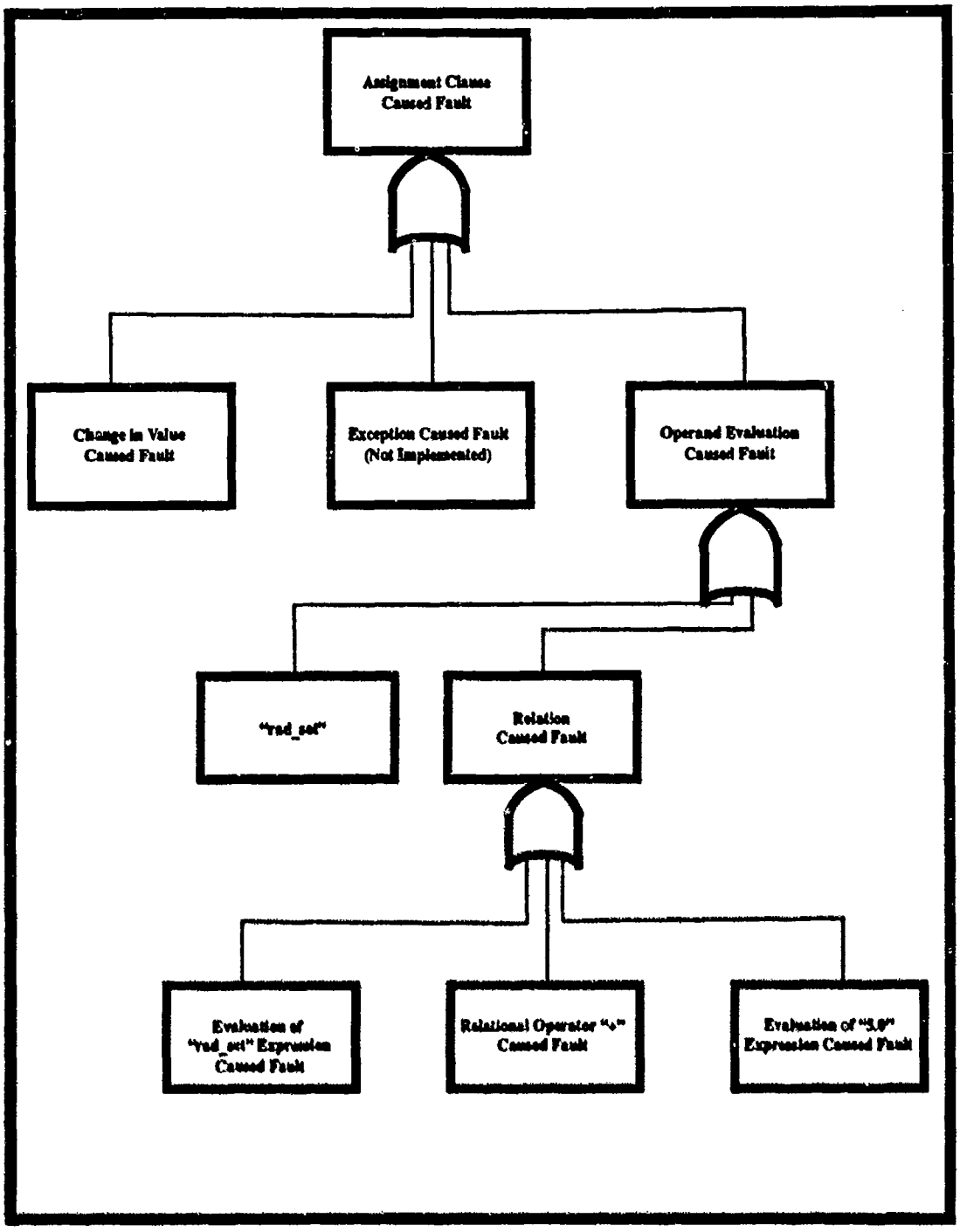


Figure 8 Assignment-Statement Template for If Clause

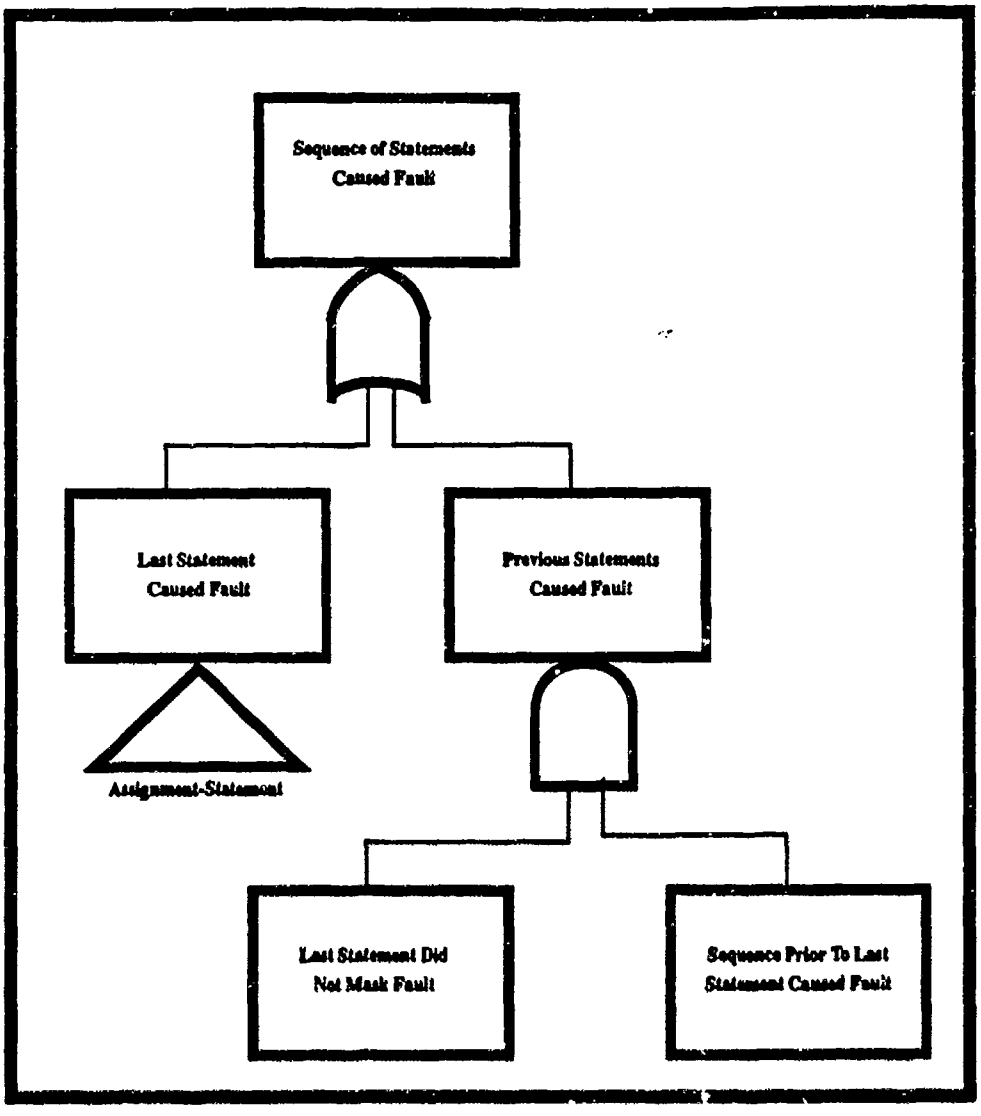


Figure 9 Sequence Of Statements Template for If Clause

The next non-terminal token to be evaluated from the if-statement production rule would be the token referring to the "elsif" clause. The elsif clause is optional. This non-terminal has three non-terminals in the production rule. These non-terminals are a non-terminal for other elsif's if applicable, a condition non-terminal, and a sequence of statements non-terminal. (See Figure 10) For the example there is only one elsif clause, therefore the other elsif non-terminal will not be executed. The elsif non-terminal is a recursive call to itself and will stop only after all elsif's have been parsed.

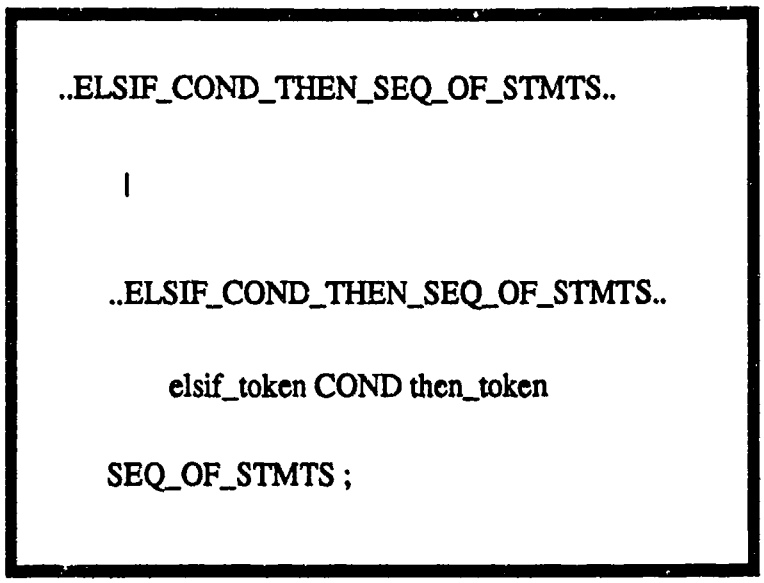


Figure 10 Production Rule for Elsif Clause

The condition for the elsif in the example is "temperature < 18.0". This condition is another simple relation and would produce a template similar to the condition template for the if condition. The sequence of statements following the elsif condition is also similar to the previous sequence of statements, another single assignment-statement. The assignment-statement is "rad_set := rad_set + 1.0;". The template generated by the template generator for this statement is identical to the previous sequence of statements except for a change in some of the values to the variables.

The condition template for the elsif and the sequence of statements template for the elsif will be passed back to the elsif non-terminal. The template generator, after the parser completes the parsing of the non-terminals in the elsif example, will use these templates to build the elsif template for the if-statement. Since there are no other elsif's in this example, the node "Next Elsif Caused Fault", which pertains to the other elsif's of the statement, will not be required. (See Figure 11)

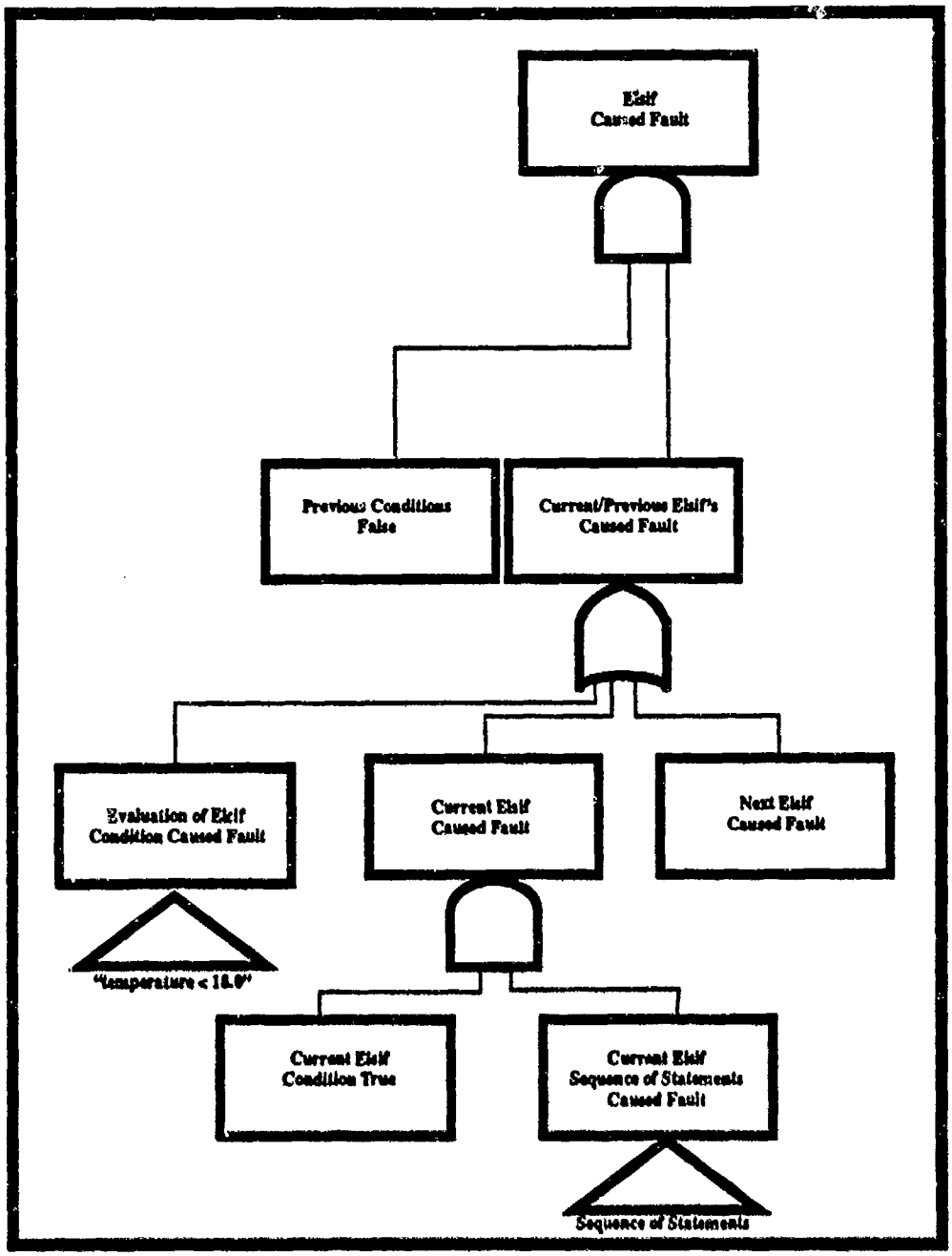


Figure 11 Elsif Template for Example

The final non-terminal token for the if-statement is the token representing the "else" clause. The else clause is optional and if it exists, it will be executed if there are no previous conditions that evaluated to true. The production rule for the else clause consist of only the sequence of statements non-terminal. (See Figure 12)

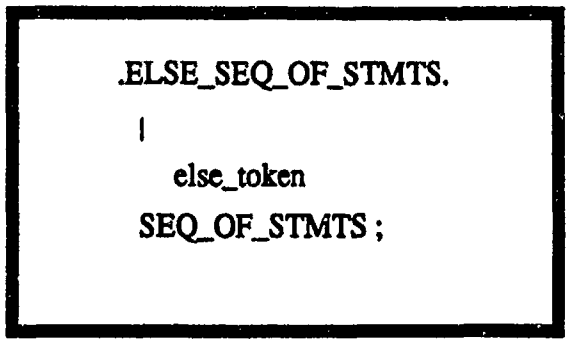


Figure 12 Production Rule for Else Clause

In parsing the sequence of statements for the example, the parser would again recognize that it consists of only one statement. This statement is the assignment-statement "rad_set := rad_set - 1.0;". Upon recognition of this fact, the template generator would create similar templates as before, excluding the condition template because the condition for the else clause is implicitly defined. The sequence of statements template will be passed back to the else non-terminal where the template generator will build the else template with the sequence of statements as a child of the template. The else template, when built, will be passed back to the if-statement non-terminal to be included in the if-statement template. (See Figure 13)

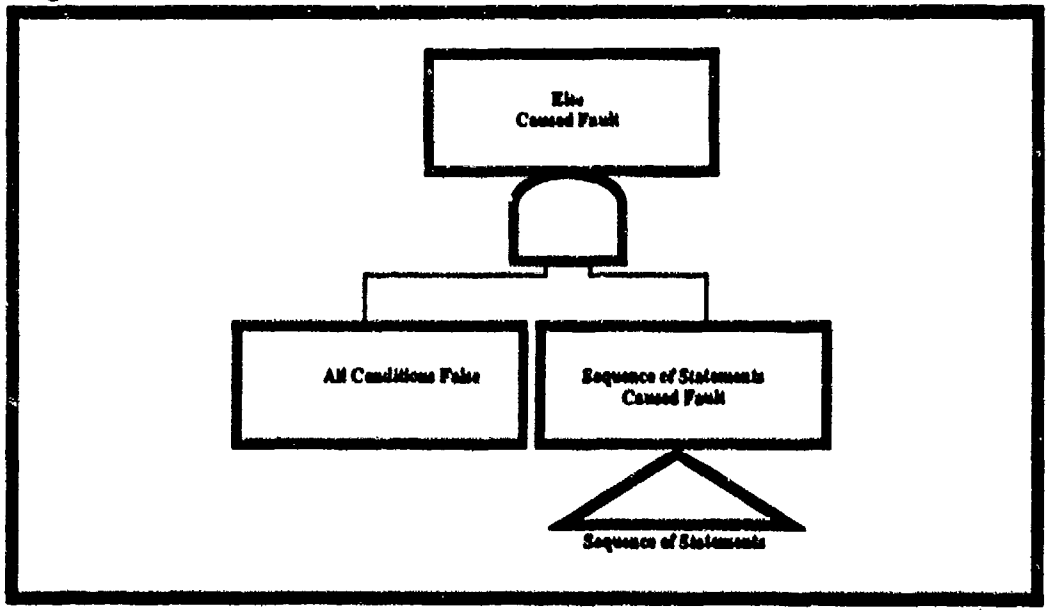


Figure 13 Else Template for Example

Once all the components of the if-statement have been parsed and the appropriate templates have been built, the template generator would be called to build the template for the if-statement. This template would be built using all of the templates that were generated by the non-terminals of the statement. (See Figure 14) Appendix C gives a more detailed version of a generic if-statement template.

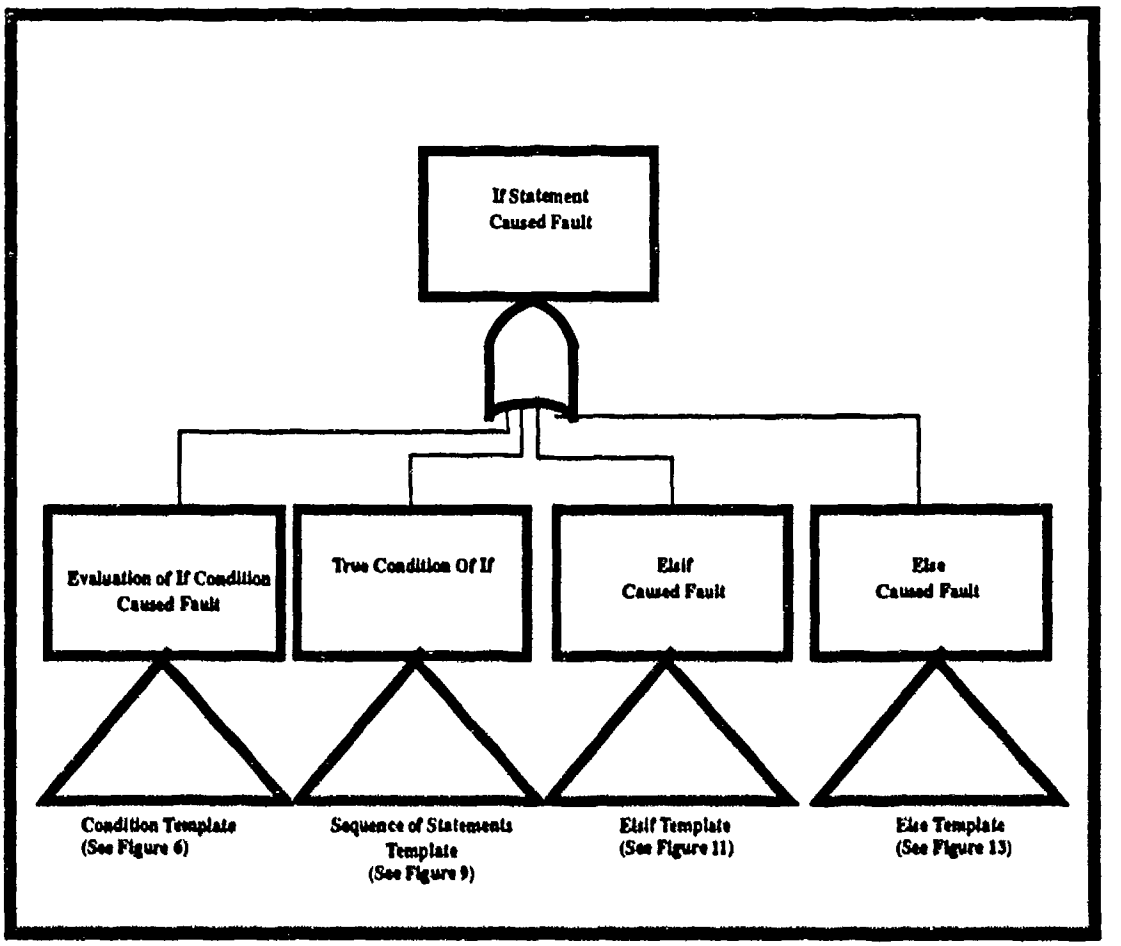


Figure 14 If-Statement Template for Example

C. THE FILE GENERATOR

The file generator is the part of the Automated Code Translation Tool that produces a FTE type file for the given input. The file generator is composed of Ada procedures that will take the generated templates and convert the data structures into FTE data structures.

Additionally, in accordance to FTE file specifications, the file generator will place the individual nodes in pre-order fashion with respect to their relationship to the root node. The file generator is embedded in the parser as actions and will be invoked when the parsing of the given input is complete. The file generator will produce a file called "NEW_FTE". This file can be loaded into FTE to display a graphical depiction of the software fault tree for the given input.

D. THE DATA STRUCTURE

The data structure for the nodes of the software fault tree was constructed using the FTE data structure as a model. This structure was used to facilitate the drawing of the software fault tree upon completion of the parsing and template building. (See Figure 15) The individual fields of the data structure that are similar to the FTE data structure have the same meaning as discussed earlier. The added fields CHILDREN_NODES and PREV_ST_PTR were added to keep track of the family ties because the nodes' relationships as parents and children are maintained by using a linked list. The other added field, OUTPUT_FLAG was not used in the prototype, but was included to allow the tool to identify structures that provide output in the program. In order to make the data structure compatible, a routine is used to equate the generated file to an FTE compatible file. This routine will remove the additional fields that were added to the data structure to facilitate the parsing. This routine is part of the file generator as discussed in the previous section.

```
type ST_LIST_NODE_PTR is access ST_LIST_NODE;

type ST_LIST_NODE is
   record
         * ST_NODE_LABEL: STRING (1..MAX_CHAR_SHORT);
         * ROOT_FAULT: STRING (1..MAX_CHAR_LONG);
         * FILE_NAME: STRING (1..MAX_CHAR_LONG);
         * START_LINE: NATURAL;
         * END_LINE: NATURAL;
         * X_COORDINATE: INTEGER;
         * Y_COORDINATE: INTEGER;
         * TYPE_NODE: NODE_TYPE;
         * TYPE_GATE: GATE_TYPE;
         * NUMBER_OF_CHILDREN: NATURAL;
           PREV_ST_PTR: ST_LIST_NODE_PTR;
           CHILDREN_NODES: ST_LIST_NODE_PTR;
           OUTPUT_FLAG: BOOLEAN;
   end record;

 *Derived from data fields in the FTE data structure
```

Figure 15 Software Fault Tree Node Data Structure

E. THE INPUT / OUTPUT

When execution begins, the tool will prompt the user for an Ada file to work with. After the user enters the file name, the tool will process the file. The prototype has limited syntax-error detection, so it is assumed that the file is syntactically correct. The output of the tool will consist of two items. The first item generated by the tool will be an on-screen display of three lists. A listing of the software fault tree will be printed to the screen first. The information provided includes the node's label and fault and the parent's label and gate, if applicable. The next list is a list of the nodes in numeric sequence. This listing was generated to facilitate the reading of the generated FTE software fault tree, which in the prototype, labels nodes by number. The last item displayed on the screen consists of a

listing of the procedures, functions, and exceptions that were located in the input file along with the first node's root fault associated with the procedure, function, or exception.

The second item produced by the tool is an ASCII file called "NEW_FTE". This file is compatible to FTE and represents the nodes produced in a pre-order listing. To display the software fault tree in FTE, FTE must be started and NEW_FTE must be in the current directory. Once FTE is started, the generated file can be loaded into FTE using the appropriate FTE commands. This will generate the software fault tree in a graphical form.

F. EXTENDED EXAMPLE

In order to see the actual code translation for a set of Ada statements performed by this tool, the tool was invoked with the file EXAMPLE.A. This file contained the procedure FAKE_OVEN_CONTROL. (See Figure 16) The procedure FAKE_OVEN_CONTROL was created to simulate an oven-control system. This procedure is by no means an accurate representation of an oven-control system, but is intended to represent source code where safety could be an important issue

```
procedure FAKE_OVEN_CONTROL(SAMPLING_DURATION : INTEGER;
                            GOAL_TEMP         : INTEGER) is

  OVEN_BUF  is access INTEGER;
  for  OVEN_BUF  use at 1234567;

  OVEN_TEMP is access INTEGER;
  for  OVEN_TEMP use at 2345671;

  LAST_CMD : INTEGER;

begin

 for SAMPLE in 1..SAMPLING_DURATION loop
   if (OVEN_TEMP < GOAL_TEMP - 10 and LAST_CMD /= 1) then
     OVEN_BUF  := 1;
     LAST_CMD  := 1;
   elsif (OVEN_TEMP < GOAL_TEMP - 10) then
     RAISE OVEN_NOT_RESPONDING;
   elsif (OVEN_TEMP > GOAL_TEMP + 10 and LAST_CMD /= 2) then
     OVEN_BUF  := 2;
     LAST_CMD  := 2;
   elsif (OVEN_TEMP > GOAL_TEMP + 10) then
     RAISE OVEN_NOT_RESPONDING;
   end if;
  for I in 1..SAMPLING_DURATION loop
    null;
   end loop;

 end loop;

end FAKE_OVEN_CONTROL;
```

Figure 16 Sample Ada Code Input

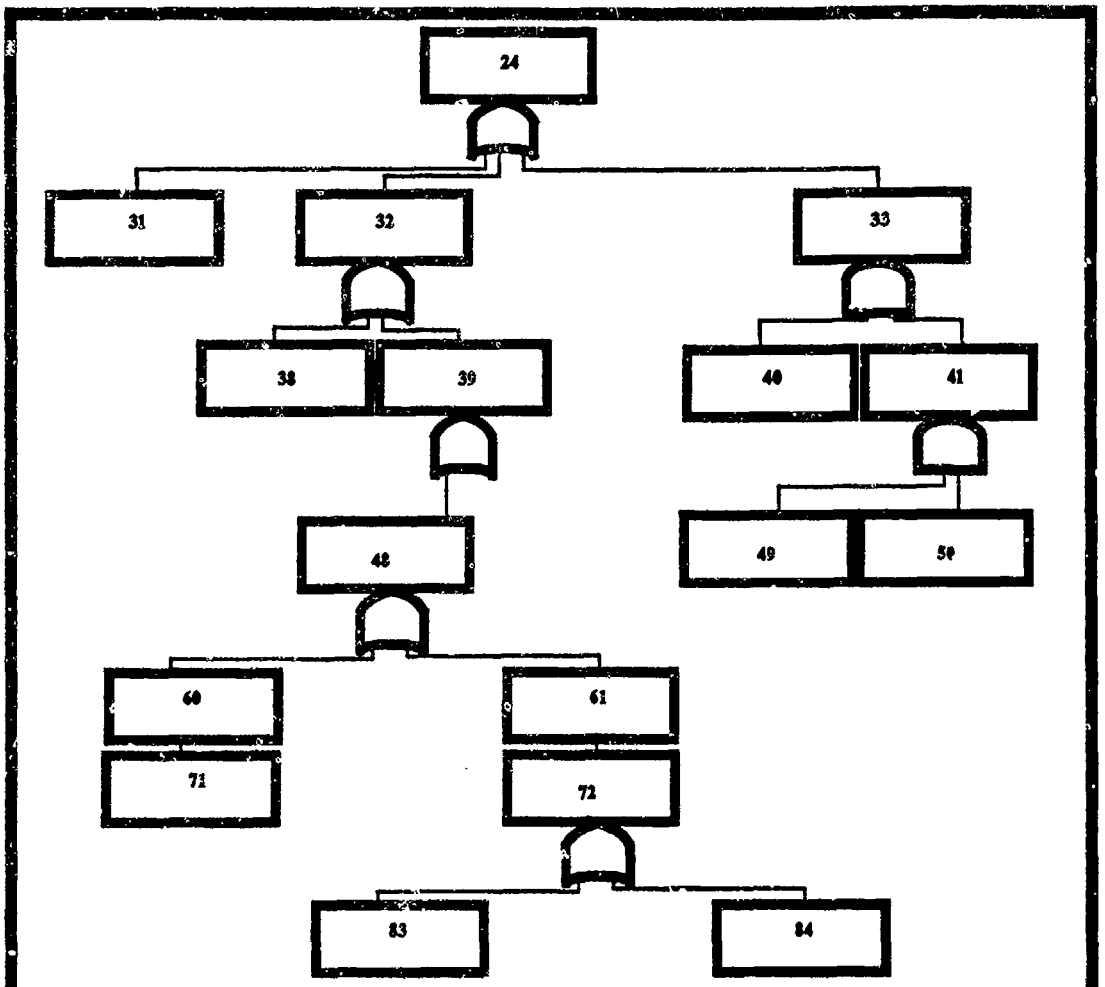
For the procedure FAKE_OVEN_CONTROL, the body of the procedure consists of a sequence of statements. A careful review of the procedure FAKE_OVEN_CONTROL shows that the sequence of statements consists of a for-loop, which is the main part of the procedure, and two embedded statements within the for-loop. The first statement within the for-loop is an if-statement that contains three elsif clauses. The second statement is another for-loop that contains the null-statement. The following discussion details how the

procedure is transformed from Ada source code to software fault tree structures. Before this discussion starts, it needs to be noted that the tool will translate the source code bottom-up, meaning that the last statement will be translated first and the first statement will be translated last. In this example there is only one statement in the outer level; however, within the outer for-loop there are two statements. The inner for-loop will be translated first and the if-statement will be translated next. The last statement to be translated will be the outer for-loop. A discussion on the translation process of the procedure follows.

In the Ada programming language, three types of loops exist, including the "for", "while", and plain loop. The Automated Code Translation Tool will build a similar template for all three loops. The major difference between the three types of loops is the iteration scheme for each loop.

If a loop caused a fault, there are three possible reasons why the loop caused the fault. First, there may exist a situation where the loop was not executed and its non-execution may cause a fault. Another possible reason may involve the evaluation of the loop condition. Finally, there may have been some situation during the Nth iteration that caused the fault. The Automated Code Translation Tool incorporates all three reasons in the loop template when the template is built.

Since the example contains a for-loop, the Automated Code Translation Tool will tailor the generic loop template accordingly. The tool will build the "Parameters Specified Caused Fault" node as a child to the "Iteration Scheme Caused Fault" node, instead of the "Evaluation of Condition Caused Fault" node which is used for a while loop. Along with this, the tool will place the sequence of statements template for the statements in the for-loop in two places. This is done to recognize the fact that the sequence of statements may have caused the fault or that the sequence of statements may have kept the loop condition true when it should have been false. In the embedded for-loop, the only statement in the sequence of statements is the null-statement. In this case, the tree representing the sequence of statements will be null. (See Figure 17)



Node Fault for Node

24 Loop Statement caused fault
31 Loop never executed
32 Loop condition evaluation caused fault
33 Nth Iteration caused fault
38 Wrong Type of loop used
39 Iteration scheme caused fault
40 Sequence of statements caused fault
41 Condition true past n-1
48 Parameter specification caused fault
49 Condition true at n-1 iteration
50 Sequence of statements kept condition true
60 Identifier specified caused fault
61 Discreet range caused fault
71 I
72 Range specified caused fault
83 1
84 SAMPLING_DURATION

Figure 17 Embedded For-Loop Template for Procedure FAKE_OVEN_CONTROL

The if-statement for the procedure FAKE_OVEN_CONTROL is similar to the example if-statement discussed earlier in the template generator section. The if-statement for the procedure, however, has three elsif clauses and no else clause. The Automated Code Translation Tool will build the if-statement template using the node "Next Elsif Caused Fault" and its associated nodes several times and not using the node "Else Caused Fault". Due to the large size of the software fault tree representing the if-statement in the procedure, the software fault tree is depicted in several figures. The figures are shown in the order that they would appear in the software fault tree and not in the order that they were generated.

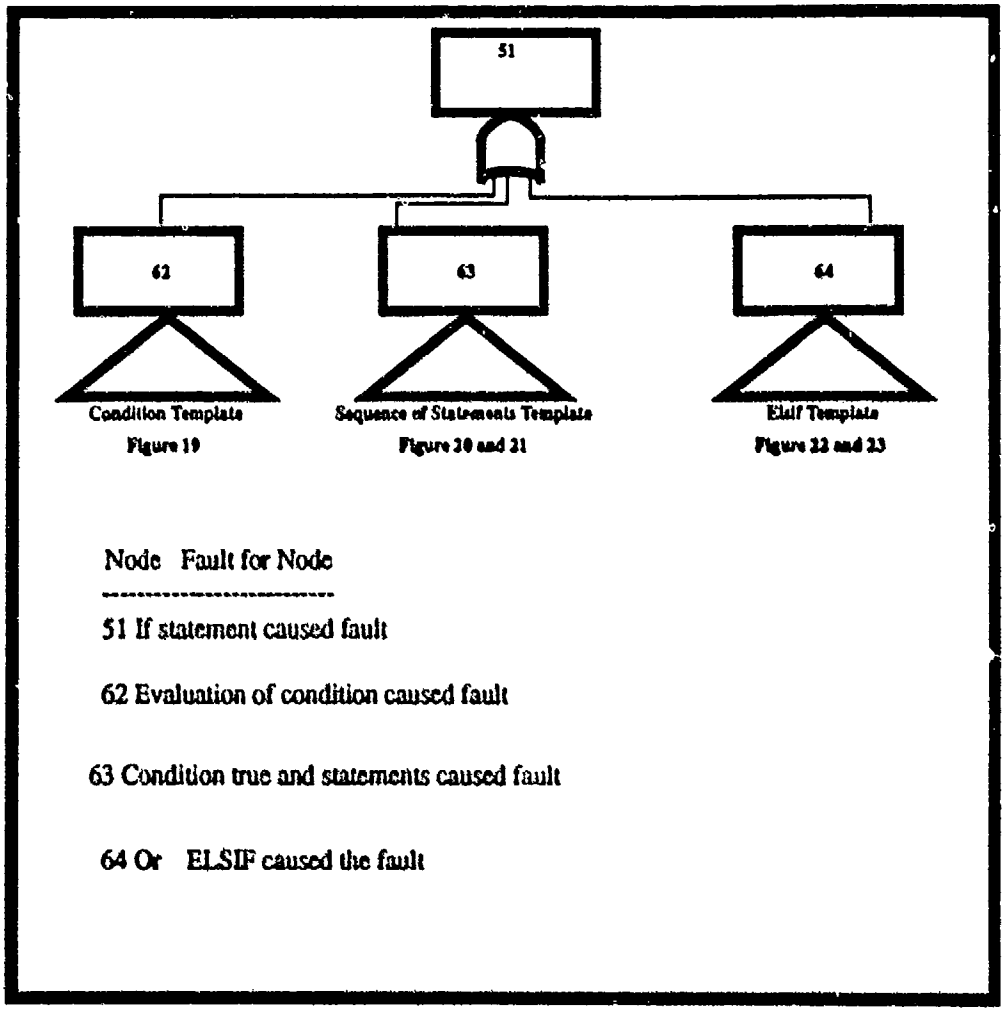
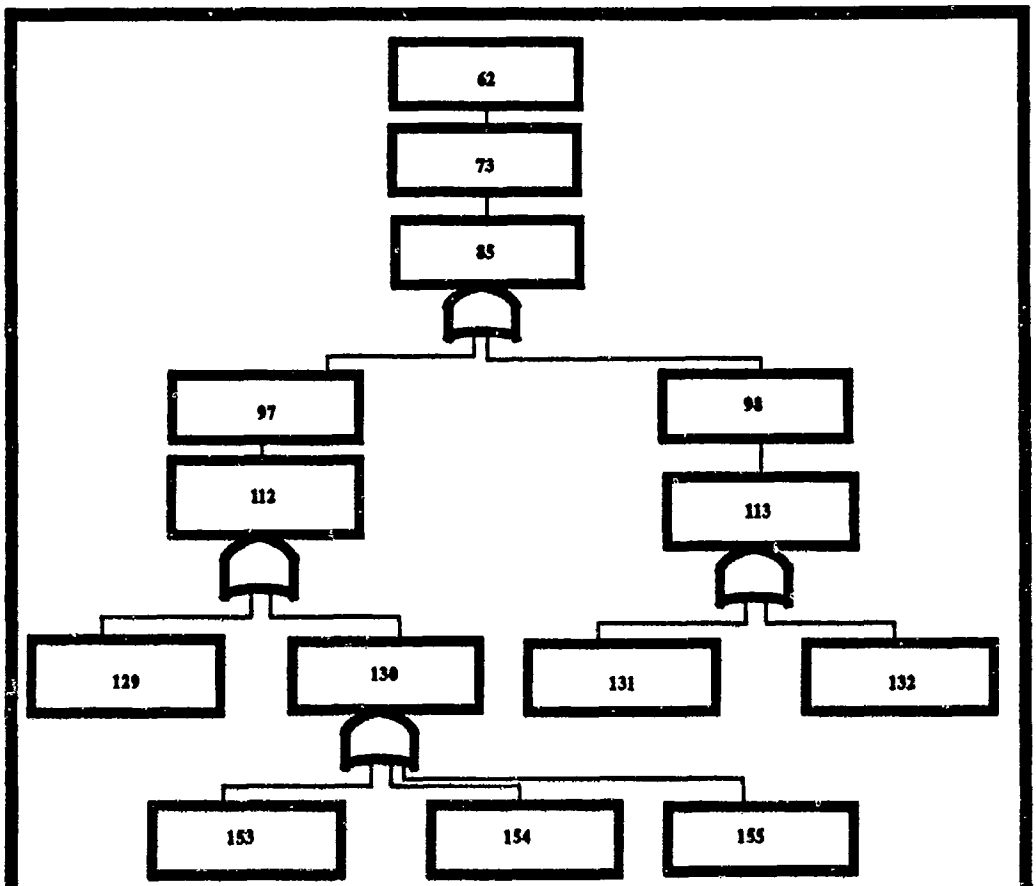


Figure 18 If-Statement Template for Procedure FAKE_OVEN_CONTROL



Node Fault for Node

62 Evaluation of condition caused fault
73 Relation caused the fault
85 And Relation caused the fault
97 Left or Prior relation caused fault
98 Right relation caused fault
112 Relation caused the fault
113 Relation caused the fault
129 OVEN_TEMP
130 Addition/Subtraction Fault
131 LAST_CMD
132 1
153 GOAL_TEMP
154 Subtraction Fault
155 10

Figure 19 Condition Template for If-Statement

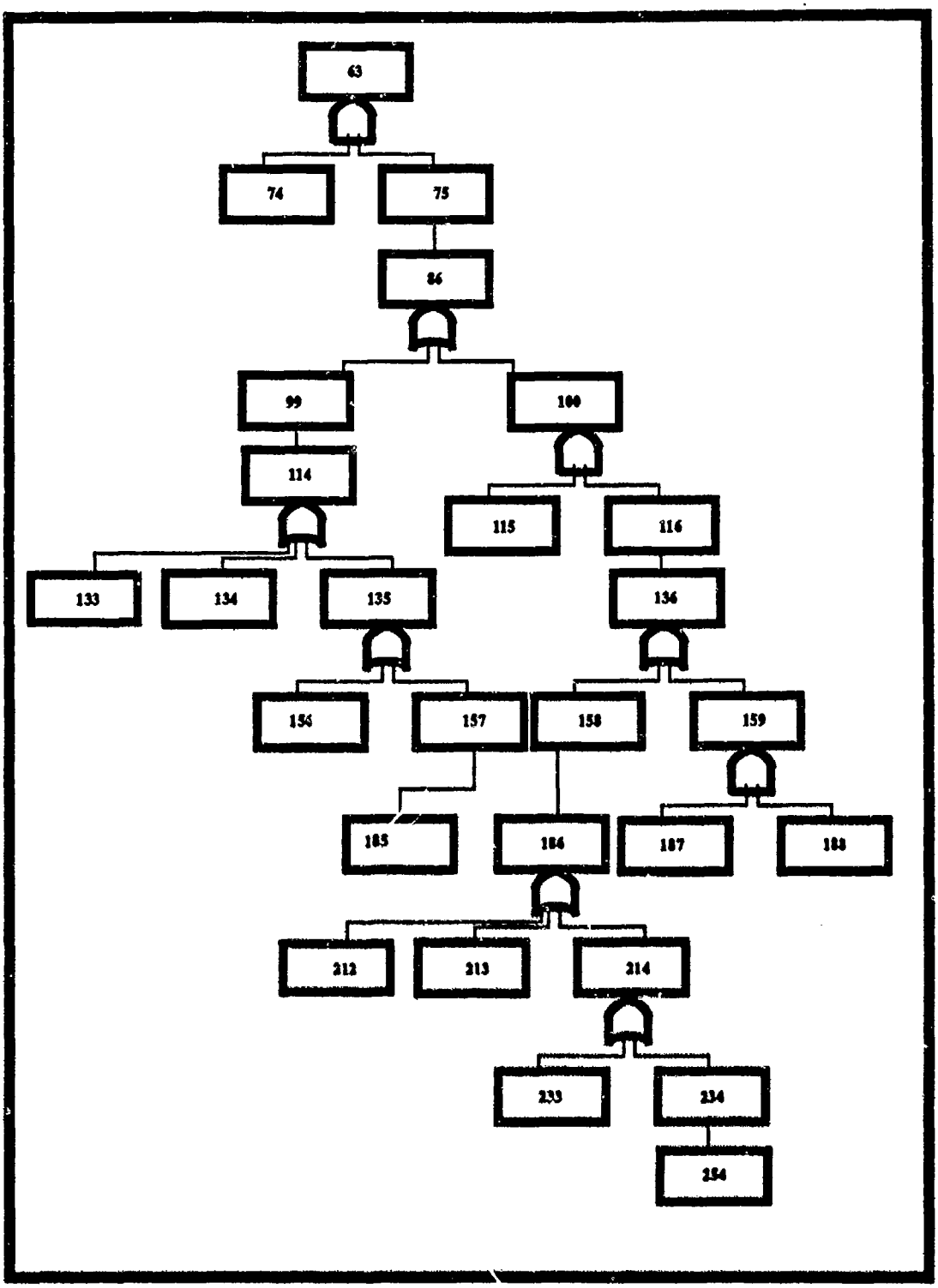


Figure 20 Sequence of Statements Template for If-Statement

Node Fault for Node

63 Condition true and statements caused fault

74 If condition true

75 If statements caused fault

86 Sequence of statements caused fault

99 Last statement caused fault

100 Previous statements caused fault

114 Assignment Statement Fault

115 Last Statement did not mask fault

116 Sequence prior to last caused fault

133 Change in values caused Fault
134 Exception causes Fault -- Not implemented
135 Operand Evaluation causes Fault

136 Sequence of statements caused fault

156 LAST_CMD

157 Relation caused the fault

158 Last statement caused fault

159 Previous statements caused fault

185 1

186 Assignment Statement Fault

187 Last Statement did not mask fault

188 And Sequence prior to last caused fault

212 Change in values caused Fault

213 Exception causes Fault -- Not implemented

214 Or Operand Evaluation causes Fault

233 OVEN_BUF

234 Relation caused the fault

254 1

Figure 21 Sequence of Statements Template for If-Statement (Continued)

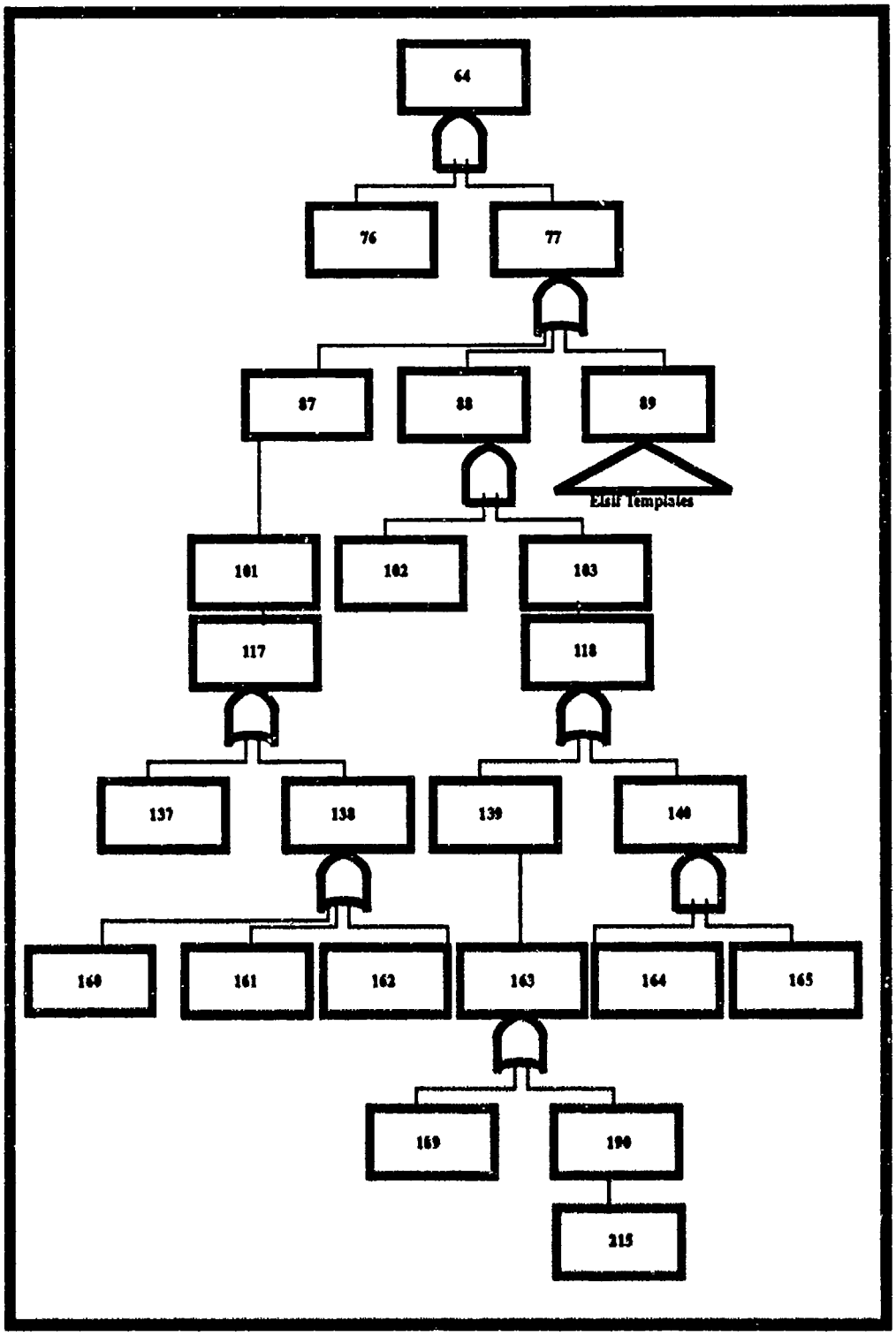


Figure 22 Elsif Template for If-Statement

Node Fault for Node

64 ELSIF caused the fault

76 Previous conditions evaluated to false

77 Current/future ELSIF caused the fault

87 Evaluation of Elsif condition caused fault

88 Current ELSIF caused the fault

89 Other ELSIF's caused the fault

101 Relation caused the fault

102 Current ELSIF condition caused the fault

103 Current ELSIF sequence of statements caused the fault

104 ELSIF caused the fault

117 Relation caused the fault

118 Sequence of statements caused fault

137 OVEN_TEMP

138 Addition/Subtraction Fault

139 Last statement caused fault

140 Previous statements caused fault

160 GOAL_TEMP

161 Addition Fault

162 10

163 Raise Statement Caused Fault

164 Last Statement did not mask fault

165 Sequence prior to last caused fault

189 Wrong Exception Raised Caused Fault

190 Exception Handler Caused Fault

215 OVEN_NOT_RESPONDING

Figure 23 Elsif Template for If-Statement (Continued)

For brevity, the templates representing the other elsif alternatives are not depicted. The templates are identical to the elsif template displayed except for a different sequence of statements for each elsif. Upon completion of the if-statement translation, the Automated Code Translation Tool builds the sequence of statements template for the outer for-loop. This template contains both the embedded for-loop and the if-statement. This template is used in the outer for-loop template representing the statements within the outer for-loop. (See Figure 24)

Basically, the template for the outer for-loop will almost be identical to the embedded for-loop. The only difference is the software fault tree representing the sequence of statements within the for-loop. For the embedded for-loop, the software fault tree was null because the statement within the loop is the null-statement. The outer for-loop contains the two statements discussed above and is represented by the sequence of statements for the outer for-loop.

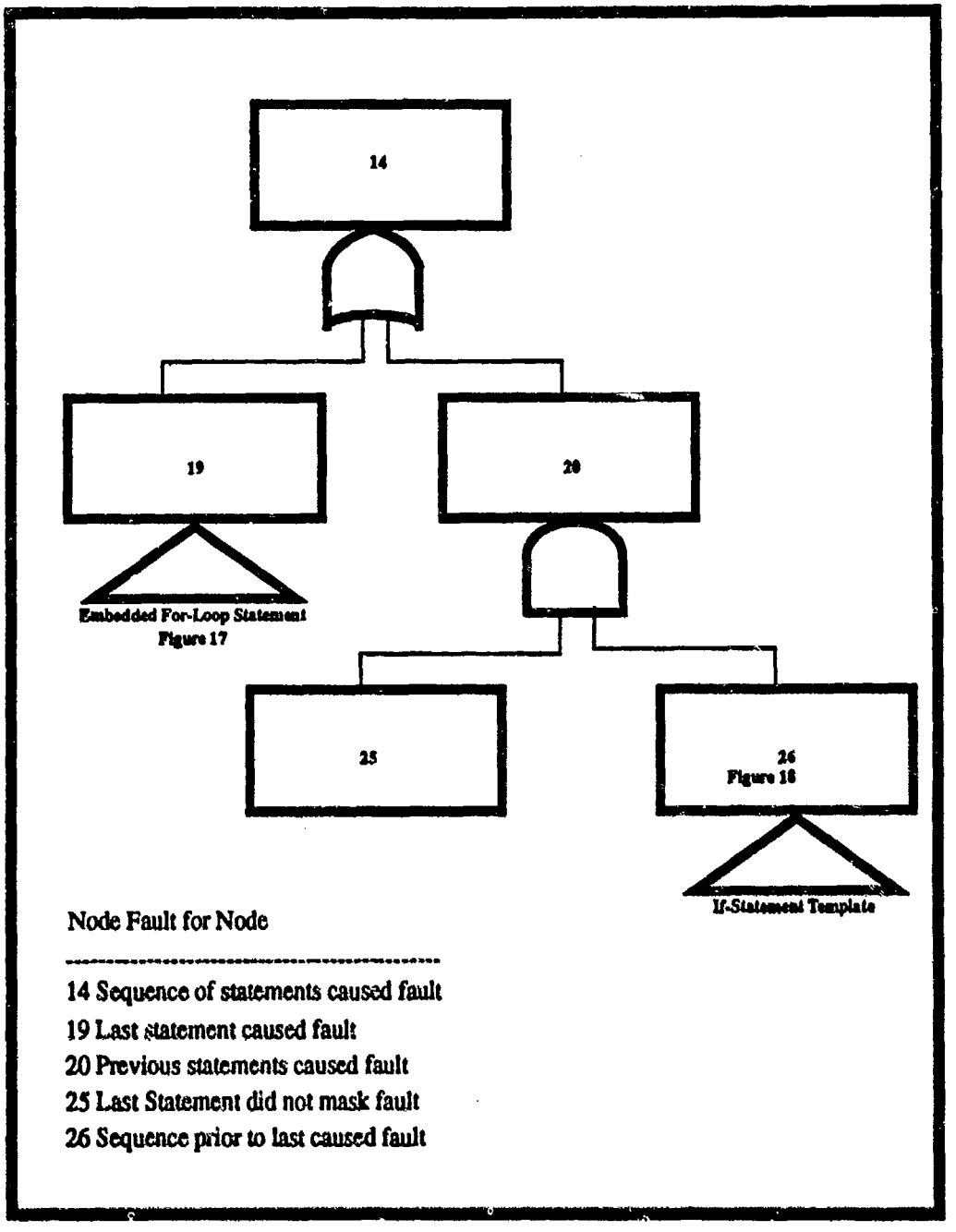
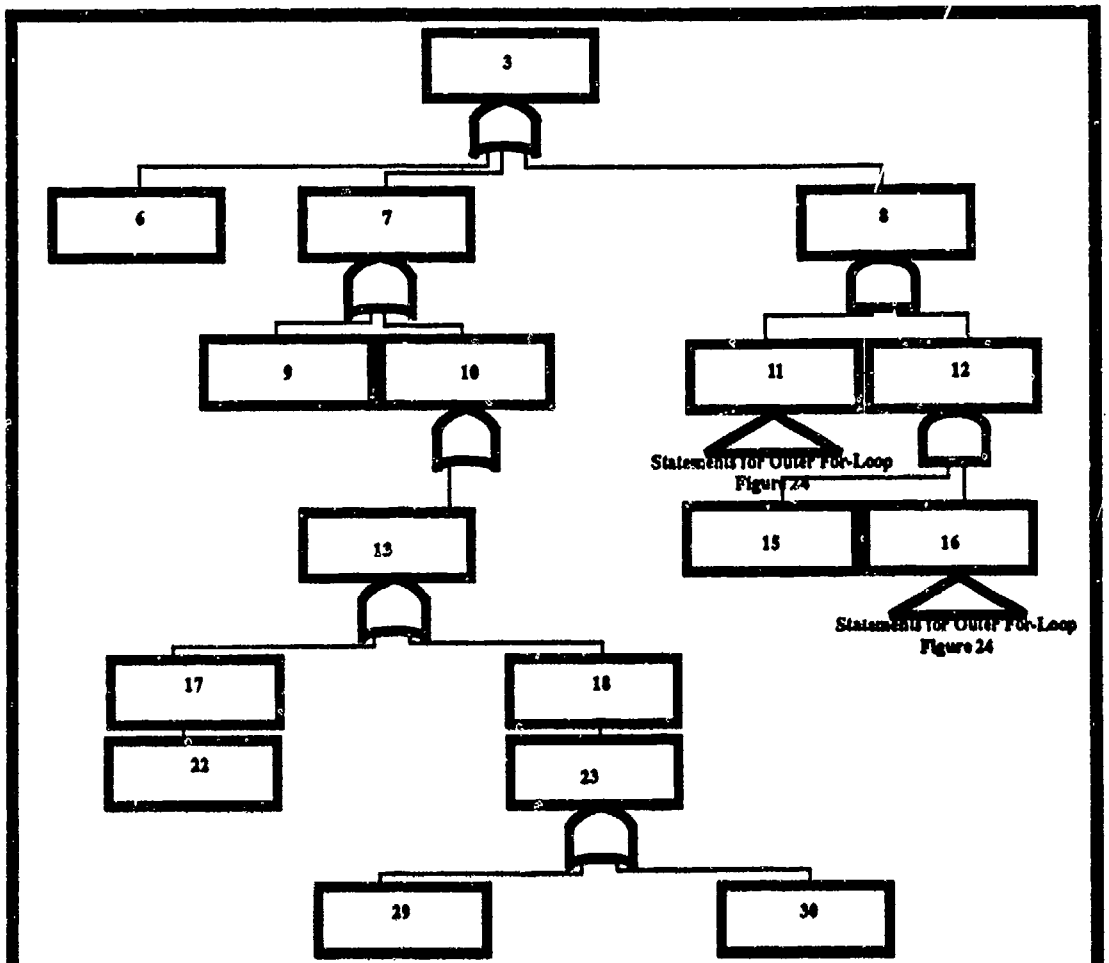


Figure 24 Sequence of Statements Template for Outer For-Loop



Node Fault for Node

3 Loop Statement caused fault
6 Loop never executed
7 Loop condition evaluation caused fault
8 Nth Iteration caused fault
9 Wrong Type of loop used
10 Iteration scheme caused fault
11 Sequence of statements caused fault
12 Condition true past n-1
13 Parameter specification caused fault
15 Condition true at n-1 iteration
16 Sequence of statements kept condition true
17 Identifier specified caused fault
18 Discreet range caused fault
22 SAMPLE
23 Range specified caused fault
29 1
30 SAMPLING_DURATION

Figure 25 Outer For-Loop Template for Procedure FAKE_OVEN_CONTROL

For the procedure FAKE_OVEN_CONTROL, the Automated Code Translation Tool will build the sequence of statements template to represent the body of the procedure. The "Last Statement Caused Fault" node will have as its child the software fault tree representing the outer for-loop. Since there is only one statement for this procedure, the "Sequence Prior to Last Caused Fault" node will be null. (See Figure 26) The result from the Automated Code Translation Tool was a software fault tree that consisted over 300 nodes. This software fault tree depicted many possible events that could cause a fault. It is now the analyst's job to select relevant events and expand the analysis accordingly.

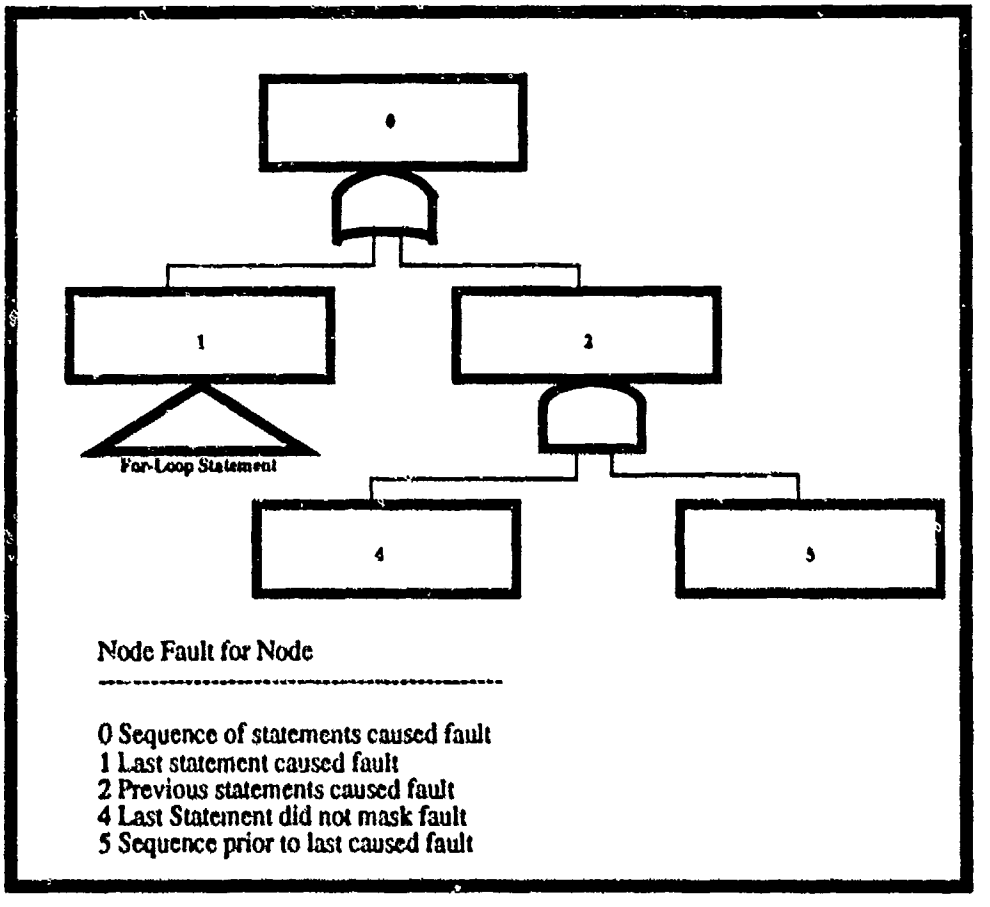


Figure 26 Sequence of Statement Template for Procedure FAKE_OVEN_CONTROL

G. FINISHING THE EXAMPLE ANALYSIS

To view the generated software fault tree in FTE, the user starts FTE and loads the file by clicking on the "LOAD" icon. FTE will now ask the user for the file to load. By typing

"NEW_FTE", the software fault tree generated by the Automated Code Translation Tool will now appear in the graphical box. The software fault tree depicted by FTE may not contain all the nodes of the NEW_FTE file because the drawing field available in FTE is limited. In order to view as many nodes as possible in the FTE drawing field, the algorithm used to generate the X and Y coordinates for the individual nodes was to place nodes on the appropriate level according to their relationship to the root node and to left justify the nodes on each level.

The Automated Code Translation Tool provides a service to the analyst. By automating the code translation, the amount of errors occurring in the translation is reduced to a minimum. Additionally, the time required by the analyst to perform this step is drastically reduced, allowing the analyst to make better use of his time. The tool, however, does not pin-point one specific fault. In order to use the generated software fault tree in SFTA, the analyst must identify a particular fault and modify the generated software fault tree accordingly.

For example in the FAKE_OVEN_CONTROL procedure there existed a for-loop that at first glance appeared to have no purpose since the loop body contained only the null-statement. In actuality, this loop was inserted as a delay to provide the control system the required time needed to perform its sensing. If this loop was removed when the code was compiled because of code optimization, the time required to sense the oven temperature would not be sufficient. This would likely cause the exception OVEN_NOT_RESPONDING to be raised. Using the generated software fault tree, the analyst could detect this fault.

Initially, the analyst must select a root fault to analyze. With the selected root fault, the analyst will eliminate the nodes from the output of the Automated Code Translation Tool that do not pertain to the selected root fault. Upon completion of this step, the analyst will need to modify the remaining nodes to ensure that they are relevant to the root fault. The end result of this process would be a software fault tree similar to the one in Figure 27. (See Figure 27)

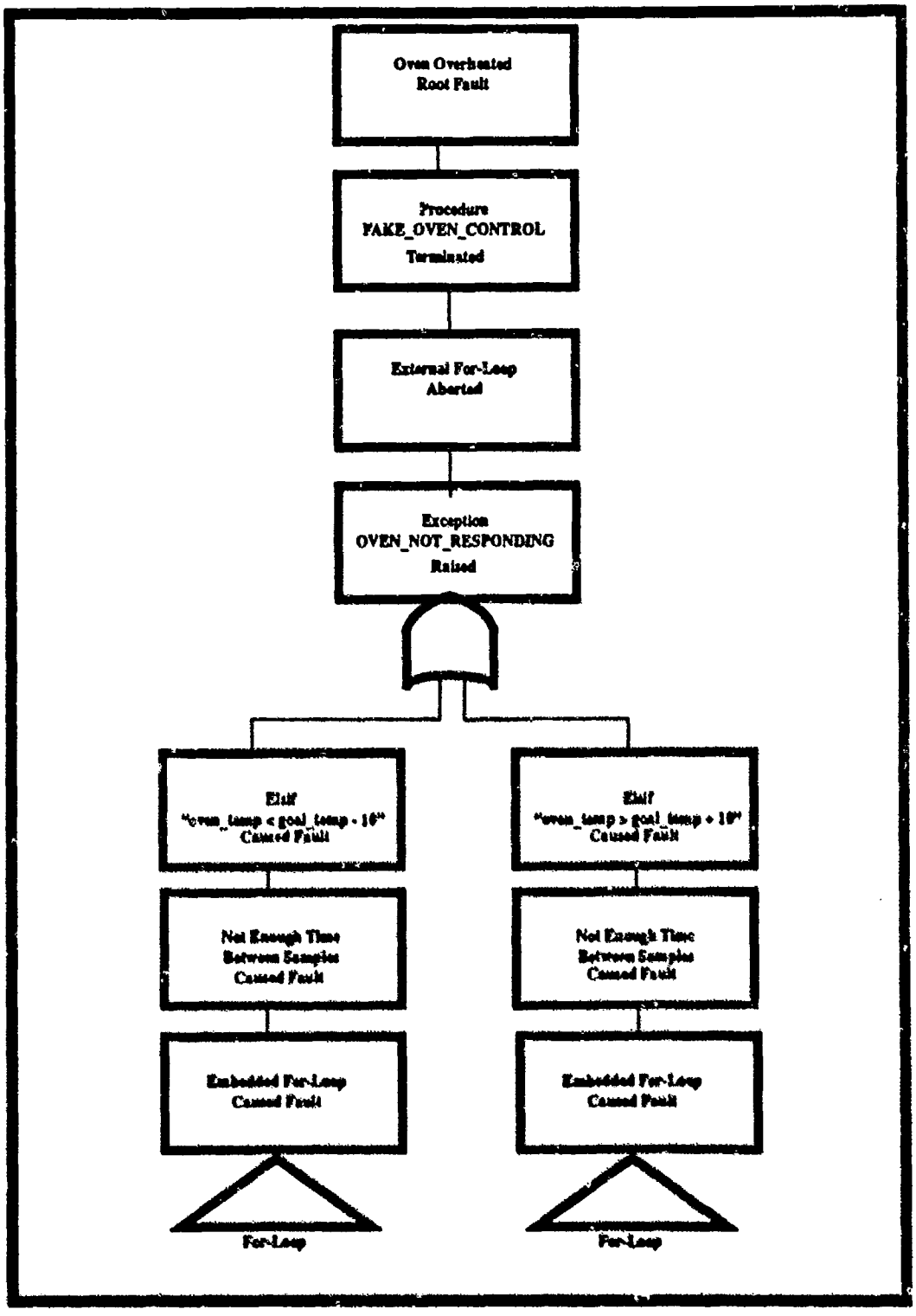


Figure 27 Modified Software Fault Tree for Procedure FAKE_OVEN_CONTROL

IV. CONCLUSIONS

A. INTRODUCTION

One question addressed by this research was to look at the possibility of automating the software fault tree analysis. Another question addressed pertained to the cost of automating the process. This chapter provides the answers to the questions. Section B summarizes the significant results of this research. Section C offers recommendations to practitioners in applying techniques of this sort. Section D concludes by giving suggestions for future research.

B. RESEARCH SUMMARY

One result of this research demonstrated that SFTA can be automated to some extent. Analyzing software in a purely manual technique is expensive because of the labor and time intensive step to traverse and translate the code. A purely manual technique may also introduce errors because humans make mistakes. The opposite solution, a totally automated process, has deficiencies because of the loss of human insight and experience of the analyst. Therefore a balance between manual and automated techniques in SFTA should be applied. The SFTA process contains a code translation step that is repetitive and labor intensive. This step is a prime candidate for automation because this step does not require much foresight since the structures of the programming language are constant. With machines inherently faster than man, using machines to perform this task can reduce the amount of time required. This, however, does not replace the need for the analyst in SFTA.

Another outcome of this study demonstrated that the cost to automate a portion of SFTA is significantly lower than the benefits that automation provides. With the existence of several tools, most notably Aflex to produce a lexical analyzer; Ayacc to produce a parser; and FTE to provide a tool to draw software fault trees, the cost to develop a tool to automatically translate code was minimal. This tool provides many benefits. An automated code translation tool provides the analyst the opportunity to use his time more efficiently

by reducing the detailed work of code translation. Additionally, the tool allows the analyst the opportunity to focus on semantics of the analysis and not the syntax of the code. Comparing the costs versus the benefits of automation, the benefits out weigh the associated costs.

C. RECOMMENDATIONS

The results of this research can help the analyst in providing safe software by facilitating the SFTA process. This thesis uses the concept of SFTA in ensuring that the software for the computer systems is safe. SFTA is not a new idea in software safety, however, the majority of research in SFTA has been performed using manual techniques. Since it has been proven in various research that tasks that require continuous and/or repetitive work is better performed by machines than man, this thesis looked at automating part of the SFTA process, specifically the code translation step. This thesis introduces a tool that removes a large percentage of the human interface. With the introduction of the Automated Code Translation Tool prototype, the tool eliminates the possible errors that could have occurred if the code translation step of SFTA was performed manually.

The Automated Code Translation Tool is a prototype to perform code translation for Ada statements. This tool works on several assumptions. Primarily, this tool is intended to provide an initial translation of the code. This translation is developed with the intention that it will be refined by the analyst. Additionally, this tool looks only at the syntax of the code. Assuming that the code is syntactically correct and that there are no overt logic errors or other errors in the code, this tool will take the input code and translate it into structures that can be used in SFTA.

This tool was developed to look at programs written in the Ada programming language. This tool should be used as a means to translate Ada statements into software fault tree structures representing the statements. The results from this tool should be followed by a thorough review by the analyst to make adjustments where necessary. This tool is a prototype and the result should not be used without an analyst.

D. FUTURE RESEARCH

There are several avenues of research that can be examined in the future as follow-up studies to this work. As a prototype, this tool looked at only a subset of Ada structures. This work can be expanded to contain the Ada structures that were not included. More specifically, the concept of tasks and exception handling need to be addressed. This tool is a stepping stone in the direction to automate code translation; however, the tool can not be complete until all of the Ada structures are implemented.

Another potential area of research might involve the interface with a graphical editor to display the software fault tree. Even though FTE provides a good graphical editor, the limitation on the drawing field poses a problem. Software fault trees tend to be large by their very nature and a graphical editor is needed to be able to depict the software fault tree in its entirety. Along with this, the algorithm used in the prototype to give the nodes X and Y coordinates was simple. To make the software fault tree more aesthetically pleasing, a more sophisticated algorithm should be developed and implemented.

Translating the output from the Automated Code Translation Tool to a usable software fault tree is another possible area of future research. An automated process to eliminate unrelated parts of the software fault tree and modify the remaining parts to be relevant to the selected root fault would reduce the amount of time necessary for software fault tree analysis and reduce the amount of possible errors caused by the human factor.

Finally, this tool provides a static code translation without any consideration to the actual fault. Future work can examine the possibility of expanding automation in SFTA by looking at the faults more dynamically as the software is working. The process of SFTA is lengthy and costly. With the introduction of automation into the process both the time to perform SFTA and the cost of SFTA can be reduced. This research has established some initial observations and steps towards automation of SFTA, but many more question are left unanswered.

APPENDIX A: AYACC AND AFLEX INPUT

A. AYACC SPECIFICATION FILE:

```
-----------------------------------------------------------------------------
-- Specification file is a modified version of the Taback and Deepak specification
-- file. The Taback and Deepak version was an adaptation of the Herman Fischer file
-- file used for Yacc.
-----------------------------------------------------------------------------
%token '&' '''' '(' ')' '*' '+' ',' '-' '.' '/' ':' ';'
%token '<' '=' '>' '|'
%token ARROW DOUBLE_DOT DOUBLE_STAR ASSIGNMENT INEQUALITY
%token GREATER_THAN_OR_EQUAL LESS_THAN_OR_EQUAL
%token LEFT_LABEL_BRACKET RIGHT_LABEL_BRACKET
%token BOX
%token ABORT_TOKEN ABS_TOKEN ACCEPT_TOKEN
            ACCESS_TOKEN
%token ALL_TOKEN AND_TOKEN ARRAY_TOKEN AT_TOKEN
%token BEGIN_TOKEN BODY_TOKEN
%token CASE_TOKEN CONSTANT_TOKEN
%token DECLARE_TOKEN DELAY_TOKEN DELTA_TOKEN
            DIGITS_TOKEN DO_TOKEN
%token ELSE_TOKEN ELSIF_TOKEN END_TOKEN ENTRY_TOKEN
            EXCEPTION_TOKEN
%token EXIT_TOKEN
%token FOR_TOKEN FUNCTION_TOKEN
%token GENERIC_TOKEN GOTO_TOKEN
%token IF_TOKEN IN_TOKEN IS_TOKEN
%token LIMITED_TOKEN LOOP_TOKEN
%token MOD_TOKEN
%token NEW_TOKEN NOT_TOKEN NULL_TOKEN
%token OF_TOKEN OR_TOKEN OTHERS_TOKEN OUT_TOKEN
%token PACKAGE_TOKEN PRAGMA_TOKEN PRIVATE_TOKEN
            PROCEDURE_TOKEN
%token RAISE_TOKEN RANGE_TOKEN RECORD_TOKEN REM_TOKEN
RENAMES_TOKEN
%token RETURN_TOKEN REVERSE_TOKEN
%token SELECT_TOKEN SEPARATE_TOKEN SUBTYPE_TOKEN
%token TASK_TOKEN TERMINATE_TOKEN THEN_TOKEN
            TYPE_TOKEN
%token USE_TOKEN
%token WHEN_TOKEN WHILE_TOKEN WITH_TOKEN
%token XOR_TOKEN
%token IDENTIFIER
%token INTEGER_LITERAL REAL_LITERAL
%token CHARACTER_LITERAL STRING_LITERAL
%token ERROR1 ERROR2 ERROR3 ERROR4 ERROR5 ERROR6
            ERROR7 ERROR8
%token ERROR9 ERROR10 ERROR11 ERROR12 ERROR13 ERROR14
```

ERROR15

```
%start compilation

%with fault_tree_generator, trace_package, traverse_pkg, text_io
%use fault_tree_generator, trace_package, traverse_pkg, text_io
{
  TAB_COUNTER   : INTEGER := 1;
  CUR_TABLE     : SYS_TABLE;
  EXCEPT_COUNTER : INTEGER := 1;
  EX_TAB        : EXCEPT_TABLE;
  FTE_FILE      : FILE_TYPE;
  OTHERS_CHECK : BOOLEAN := FALSE;
  type stype_var is (int, nde);
  type yystype (form:stype_var := int) is
   record
    case form is
     when int => int_t :  integer;
     when nde => node_t:  st_list_node_ptr;
    end case;
   end record;
}
 %%
set_others_true : {OTHERS_CHECK := TRUE;} ;
set_others_false : {OTHERS_CHECK := FALSE;} ;

prag   :
       PRAGMA_TOKEN IDENTIFIER .arg_ascs ';' ;
arg_asc    :
       expr
  |    IDENTIFIER ARROW expr ;
-- *** Added *** --
numeric_literal
       : REAL_LITERAL
       | INTEGER_LITERAL

;basic_d :
       object_d
  |    ty_d          | subty_d
  |    subprg_d  | pkg_d
  |    task_d        | gen_d
  |    excptn_d  | gen_inst
  |    renaming_d
  |    number_d
  |    error ';' ;

object_d :
    idents ':'        subty_ind ._ASN_expr. ';'
|   idents ':' CONSTANT_TOKEN subty_ind ._ASN_expr. ';'
|   idents ':'        c_arr_def ._ASN_expr. ';'
|   idents ':' CONSTANT_TOKEN c_arr_def ._ASN_expr. ';' ;
```

```
number_d :
        idents ':' CONSTANT_TOKEN ASSIGNMENT expr ';' ;

idents  : IDENTIFIER ...ident..;

ty_d :
        full_ty_d
  |     incomplete_ty_d
  |     priv_ty_d ;

full_ty_d :
      TYPE_TOKEN IDENTIFIER          IS_TOKEN ty_def ';'
  |   TYPE_TOKEN IDENTIFIER discr_part IS_TOKEN ty_def ';' ;

ty_def :
        enum_ty_def     | integer_ty_def
  |     real_ty_def     | array_ty_def
  |     rec_ty_def      | access_ty_def
  |     derived_ty_def ;

subty_d :
        SUBTYPE_TOKEN IDENTIFIER IS_TOKEN subty_ind ';' ;

subty_ind :ty_mk .constrt. ;

ty_mk : expanded_n ;

constrt :
        rng_c
  |     fltg_point_c
  |     fixed_point_c
  |     aggr ;

derived_ty_def : NEW_TOKEN subty_ind ;

rng_c : RANGE_TOKEN rng  {$$ := $2;} ;

rng :
        name    {$$ := (form    => nde,
                        node_t => BUILD_RANGE_NAME ($1.node_t)); }
  |     sim_expr DOUBLE_DOT sim_expr{
                $$ := (form     => nde,
                        node_t => BUILD_RNG ($1.node_t,
                                        $3.node_t));} ;

enum_ty_def     :
        '(' enum_lit_spec
                        ...enum_lit_spec.. ')' ;

enum_lit_spec : enum_lit ;
```

```
enum_lit : IDENTIFIER | CHARACTER_LITERAL ;

integer_ty_def : rng_c ;

real_ty_def    :
        fltg_point_c | fixed_point_c ;

fltg_point_c :
        fltg_accuracy_def .rng_c. ;

fltg_accuracy_def :
        DIGITS_TOKEN sim_expr ;

fixed_point_c:
        fixed_accuracy_def .rng_c. ;

fixed_accuracy_def :
        DELTA_TOKEN sim_expr ;

array_ty_def    :
        uncnstrnd_array_defl c_arr_def ;
uncnstrnd_array_def:
     ARRAY_TOKEN '(' idx_subty_def ...idx_subty_def.. ')' OF_TOKEN
                                        subty_ind ;

c_arr_def    :
        ARRAY_TOKEN idx_c OF_TOKEN subty_ind ;

idx_subty_def   :   name RANGE_TOKEN BOX ;

idx_c : '(' dscr_rng ...dscr_rng.. ')' ;

dscr_rng:
        rng          {$$ := (form   => nde,
                      node_t => BUILD_DESCR_RNG_1 ($1.node_t)); }
  |     name rng_c{$$ := (form   => nde,
                      node_t => BUILD_DESCR_RNG_2 ($1.node_t,
                                        $2.node_t)); } ;

rec_ty_def :
        RECORD_TOKEN
          cmpons
        END_TOKEN RECORD_TOKEN ;

cmpons:
        ..prag.. ..cmpon_d.. cmpon_d ..prag..
  |     ..prag.. ..cmpon_d.. variant_part ..prag..
  |    ..prag.. NULL_TOKEN ';' ..prag.. ;

cmpon_d :
   idents ':' cmpon_subty_def ._ASN_expr. ';' ;
```

```
cmpon_subty_def    : subty_ind ;

discr_part :
     '(' discr_spec ...discr_spec.. ')' ;

discr_spec :
       idents ':' ty_mk ._ASN_expr. ;

variant_part :
       CASE_TOKEN sim_n IS_TOKEN
           ..prag.. variant ..variant..
     END_TOKEN CASE_TOKEN ';' ;

variant :
       WHEN_TOKEN choice ..or_choice.. ARROW
         cmpons ;

choice : sim_expr                  $$ := $1; }
       | name rng_c                ($$ := (form   => nde,
                                  node_t => BUILD_CHOICE_2
                                      ($1.node_t,
                                      $2.node_t)); }
       | sim_expr DOUBLE_DOT sim_expr
                           {$$ := (form   => nde,
                           node_t => BUILD_CHOICE_3
                                      ($1.node_t,
                                       $3.node_t)); }
       | OTHERS_TOKEN set_others_true
       | error;

access_ty_def : ACCESS_TOKEN subty_ind ;

incomplete_ty_d :
     TYPE_TOKEN IDENTIFIER ';'
  |  TYPE_TOKEN IDENTIFIER discr_part ';' ;

decl_part :
         ..basic_decl_item..
       | ..basic_decl_item.. body ..later_decl_item.. ;

basic_decl_item :
       basic_d
  |    rep_cl | use_cl;

later_decl_item : body
  |    subprg_d        | pkg_d
  |    task_d          | gen_d
  |    use_cl          | gen_inst  ;

body : proper_body
```

```
   | body_stub ;

proper_body :
        subprg_body
      | pkg_body
      | task_body;

name : sim_n {$$ := $1;}
   | CHARACTER_LITERAL
   | op_symbol
   | idxed_cmpon {$$ := $1;}
   | selected_cmpon
   | attribute ;

sim_n :IDENTIFIER {$$ :=
                        (form  => nde,
                        node_t => NAME_TEMPLATE (YYText)); } ;

prefix : name {$$ := $1;} ;

idxed_cmpon :
       prefix aggr {$$ :=
                        (form  => nde,
                        node_t => INDEX_COMP_TEMPLATE
                                ($1.node_t,
                                $2.node_t)); } ;

selected_cmpon : prefix '.' selector ;

selector : sim_n
  | CHARACTER_LITERAL | op_symbol | ALL_TOKEN ;

attribute : prefix "'" attribute_designator ;

attribute_designator :
       sim_n
  |    DIGITS_TOKEN
  |    DELTA_TOKEN
  |    RANGE_TOKEN;

aggr :
     '(' cmpon_asc ...cmpon_asc.. ')' {$$ := $2;} ;

cmpon_asc      :
       expr {$$ := $1;}
  |    choice ..or_choice.. ARROW expr
  |    sim_expr DOUBLE_DOT sim_expr
  |    name rng_c ;

expr :
       rel_AND_rel..     {$$ := $1;}
```

```
    |   rel..AND__THEN__rel..    {$$ := $1;}
    |   rel..OR__rel..                  {$$ := $1;}
    |   rel..OR__ELSE__rel..     {$$ := $1;}
    |   rel..XOR__rel..                 {$$ := $1;} ;

rel :
        sim_expr .relal_op__sim_expr. {$$ := (form  => nde,
                                node_t => BUILD_REL ($1.node_t,
                                                $2.node_t));}
    |   sim_expr.NOT.IN__rng_or_sim_expr.NOT.IN__ty_mk ;

sim_expr :
        .unary_add_op.term..binary_add_op__term.. {$$ := $1;} ;

term : factor..mult_op__factor.. {$$ := $1;} ;

factor : pri ._EXP___pri. {$$ := $1;}
    | ABS_TOKEN pri   {$$ := $2;}
    | NOT_TOKEN pri   {$$ := $2;};

pri :
        numeric_literal {$$ := (form  => nde,
                        node_t => VALUE_TEMPLATE (YYText));}
    | NULL_TOKEN     {$$ := (form  => nde,
                        node_t => null);}
    |   allocator
    | qualified_expr
    |   name {$$ := (form  => nde,
                node_t => CHECK_FUNCTION ($1.NODE_T,
                                CUR_TABLE,
                                TAB_COUNTER)); }
    |   aggr {$$ := $1;} ;

relal_op : '='
        | INEQUALITY
        | '<'
        | LESS_THAN_OR_EQUAL
        | '>'
        | GREATER_THAN_OR_EQUAL ;

binary_add_op : '+' {$$ := (form  => nde,
                        node_t => (ADD_FAULT_TEMPLATE));}
        | '-' {$$ := (form  => nde,
                        node_t => (SUB_FAULT_TEMPLATE));}
        | '&';

unary_add_op : '+' | '-' ;

mult_op : '*' {$$ := (form  => nde,
                        node_t => null);}
    | '/' {$$ :=        (form  => nde,
```

```
                        node_t => DIVISION_BY_ZERO_TEMPLATE);}
    | MOD_TOKEN
    | REM_TOKEN ;

qualified_expr:
      ty_mkaggr_or_ty_mkPexprP_ ;

allocator :
      NEW_TOKEN ty_mk
  |   NEW_TOKEN ty_mk aggr
  |   NEW_TOKEN ty_mk '" aggr ;

seq_of_stmts: ..prag.. stmt ..stmt..
{$$ :=          (form  => nde,
                node_t => STATEMENTS
                        ($3.NODE_T,
                         $2.NODE_T));};

stmt :
      ..label.. sim_stmt     {$$ := $2;}
  | ..label.. compound_stmt {$$ := $2;}
  | error  ';' ;

sim_stmt :null_stmt     {$$ := $1;}
      |assignment_stmt {$$ := $1;}
      |exit_stmt
      |return_stmt
      |goto_stmt
      |delay_stmt
      |abort_stmt
      |raise_stmt{$$ := $1;}
      |code_stmt
      |name ';'
            {$$ := (form  => nde,
                  node_t => BUILD_PROC_CALL
                        ($1.node_t,
                        CUR_TABLE,
                        TAB_COUNTER)); } ;

compound_stmt :
      if_stmt    {$$ := $1;}
  |   case_stmt  {$$ := $1;}
  |   loop_stmt  {$$ := $1;}
  |   block_stmt {$$ := $1;}
  |   accept_stmt
  |   select_stmt ;

label : LEFT_LABEL_BRACKET sim_n RIGHT_LABEL_BRACKET ;

null_stmt : NULL_TOKEN ';'
      {$$ := (form  => nde, node_t => null);};
```

```
assignment_stmt : name ASSIGNMENT expr ';'
 {$$ := (form  => nde, node_t => ASSIGNMENT_TEMPLATE
                                 ($1.node_t,
                                 $3.node_t));};

if_stmt :
        IF_TOKEN cond THEN_TOKEN
          seq_of_stmts
        ..ELSIF__cond__THEN__seq_of_stmts..
        .ELSE__seq_of_stmts.
        END_TOKEN IF_TOKEN ';'
        {$$ := (form  => nde, node_t => IF_STATEMENT_TEMPLATE
                                 ($2.node_t,
                                 $4.node_t,
                                 $5.node_t,
                                 $6.node_t));};

cond : expr {$$ := $1;};

case_stmt:
        CASE_TOKEN expr IS_TOKEN
          case_stmt_alt..case_stmt_alt..
        END_TOKEN CASE_TOKEN ';'
          set_others_false
        {$$ := (form  => nde, node_t => BUILD_CASE_STMT_TEMPLATE
                                 ($2.node_t,
                                 $4.node_t)); } ;

case_stmt_alt :
        WHEN_TOKEN choice ..or_choice.. ARROW
          seq_of_stmts
        {$$ := (form  => nde, node_t => BUILD_CASE_STMT_ALT
                                 ($2.node_t,
                                 $3.node_t,
                                 $5.node_t,
                                 OTHERS_CHECK )); } ;

loop_stmt:
        .sim_nC.
          .iteration_scheme. LOOP_TOKEN
                seq_of_stmts
        END_TOKEN LOOP_TOKEN .sim_n. ';'
        {$$ := (form  => nde, node_t => BUILD_LOOP_STMT
                ($2.node_t, $4.node_t)); } ;

iteration_scheme
        : WHILE_TOKEN cond
{$$ := (form  => nde, node_t => BUILD_WHILE_ITERATION
                                 ($2.node_t)); }
        | WHILE_TOKEN error
```

```
      | FOR_TOKEN loop_prm_spec
{$$ := (form  => nde,  node_t => BUILD_FOR_ITERATION ($2.node_t)); }
      | FOR_TOKEN error

loop_prm_spec :
      IDENTIFIER
{$$ := (form  => nde, node_t => NAME_TEMPLATE (YYTEXT));}

    IN_TOKEN
    .REVERSE.
    dscr_mg
{$$ := (form  => nde  node_t => BUILD_LOOP_PRM_SPEC
                                ($2.node_t,  $4.node_t,  $5.node_t));} ;

block_stmt :
      .sim_nC.
        .DECLARE__decl_part.
        BEGIN_TOKEN
              seq_of_stmts
        .EXCEPTION__excptn_handler..excptn_handler...
      END_TOKEN .sim_n. ';'
      {$$ := (form  => nde, node_t => BUILD_BLOCK_STMT
                                ($1.NODE_T,
                                $4.NODE_T,
                                $5.NODE_T)); } ;

exit_stmt:
    EXIT_TOKEN .expanded_n. .WHEN__cond. ';' ;

return_stmt : RETURN_TOKEN .expr. ';' ;

goto_stmt : GOTO_TOKEN expanded_n ';' ;

subprg_d : subprg_spec ';' ;

subprg_spec :
      PROCEDURE_TOKEN
    IDENTIFIER
{$$ := (form  => nde, node_t => NAME_TEMPLATE(YYTEXT)); }
    .fml_part.
    {$$ := $3;}

  |   FUNCTION_TOKEN
    designator
    .fml_part.
    RETURN_TOKEN
    ty_mk
    {$$ := $2; } ;

designator : IDENTIFIER
{$$ := (form  => nde, node_t => NAME_TEMPLATE(YYTEXT)); }
```

```
        op_symbol    {$$ := $1; } ;

op_symbol : STRING_LITERAL
{$$ := (form  => nde,  node_t => NAME_TEMPLATE(YYTEXT)); } ;

fml_part :
     '(' prm_spec .._.prm_spec.. ')' ;

prm_spec :
        idents ':' mode ty_mk ._ASN_expr. ;

mode : .IN. | IN_TOKEN OUT_TOKEN | OUT_TOKEN;

subprg_body :
        subprg_spec IS_TOKEN
          .decl_part.
        BEGIN_TOKEN
          seq_of_stmts
        .EXCEPTION__excptn_handler..excptn_handler...
     END_TOKEN .designator. ';'
     {CUR_TABLE(TAB_COUNTER).P_F_NAME :=
RETURN_ROOT_FAULT($1.NODE_T);
CUR_TABLE(TAB_COUNTER).STMT_PTR := $5.NODE_T;
TAB_COUNTER                  := TAB_COUNTER + 1;
$$ := $5;};

pkg_d : pkg_spec ';' ;

pkg_spec :
        PACKAGE_TOKEN IDENTIFIER IS_TOKEN
          ..basic_decl_item..
        .PRIVATE..basic_decl_item...
        END_TOKEN .sim_n.  ;

pkg_body :
        PACKAGE_TOKEN BODY_TOKEN sim_n IS_TOKEN
          .decl_part.
          .BEGIN__seq_of_stmts.EXCEPTION!__excptn_handler..excptn_handler...
        END_TOKEN .sim_n. ';'  ;

priv_ty_d :
        TYPE_TOKEN IDENTIFIER          IS_TOKEN .LIMITED. PRIVATE_TOKEN ';'
    | TYPE_TOKEN IDENTIFIER discr_part IS_TOKEN .LIMITED. PRIVATE_TOKEN ';' ;

use_cl : USE_TOKEN expanded_n ..expanded_n.. ';' ;

renaming_d :
     idents ':' ty_mk     RENAMES_TOKEN name ';'
  |  idents ':' EXCEPTION_TOKEN     RENAMES_TOKEN expanded_n ';'
  |  PACKAGE_TOKEN IDENTIFIER       RENAMES_TOKEN expanded_n ';'
  |  subprg_spec RENAMES_TOKEN name ';'  ;
```

```
task_d : task_spec ';' ;

task_spec :
        TASK_TOKEN .TYPE. IDENTIFIER
          .IS..ent_d_.rep_cl_END.sim_n.
          ;

task_body :
        TASK_TOKEN BODY_TOKEN sim_n IS_TOKEN
          .decl_part.
        BEGIN_TOKEN
          seq_of_stmts
        .EXCEPTION__excptn_handler..excptn_handler...
      END_TOKEN .sim_n. ';' ;

ent_d :
        ENTRY_TOKEN IDENTIFIER .fml_part. ';'
  |  ENTRY_TOKEN IDENTIFIER '(' dscr_rng ')' .fml_part. ';' ;

ent_call_stmt :
      ..prag.. name ';' ;

accept_stmt :
        ACCEPT_TOKEN sim_n .Pent_idx_P..fml_part.
        .DO__seq_of_stmts__END.sim_n.. ';' ;

ent_idx :expr ;

delay_stmt : DELAY_TOKEN sim_expr ';' ;

selec_stmt :selec_wait
  |     condal_ent_call| timed_ent_call ;

selec_wait:
        SELECT_TOKEN
          select_alt
          ..OR__select_alt..
          .ELSE__seq_of_stmts.
      END_TOKEN SELECT_TOKEN ';' ;

select_alt :
        .WHEN__condARROW.selec_wait_alt ;

selec_wait_alt : accept_alt
  |     delay_alt | terminate_alt ;

accept_alt :
        accept_stmt.seq_of_stmts. ;

delay_alt :
```

```
        delay_stmt.seq_of_stmts. ;

terminate_alt : TERM_stmt  ;

condal_ent_call:
      SELECT_TOKEN
        ent_call_stmt
        .seq_of_stmts.
      ELSE_TOKEN
        seq_of_stmts
    END_TOKEN SELECT_TOKEN ';' ;

timed_ent_call :
      SELECT_TOKEN
        ent_call_stmt
        .seq_of_stmts.
      OR_TOKEN
        delay_alt
    END_TOKEN SELECT_TOKEN ';' ;

abort_stmt : ABORT_TOKEN name ...name.. ';' ;

compilation :..compilation_unit..
          {-- Code to print out the system table of proc and funct
          NEW_PAGE;
          NEW_LINE;
          if TAB_COUNTER - 1 > 0 then
           PUT    ("The number of procedures/functions on the table are ");
           PUT(TAB_COUNTER - 1, width => 5);
           NEW_LINE(2);
           PUT_LINE("The procedures/functions and their root faults on the table ");
           PUT_LINE("are the following:  ");
           PUT_LINE("-------------------------------------------------------");
           NEW_LINE;
           for INDEX in 1..TAB_COUNTER - 1 loop
            PUT_LINE(CUR_TABLE(INDEX).P_F_NAME);
            PUT_LINE(RETURN_ROOT_FAULT(CUR_TABLE(INDEX).STMT_PTR));
            NEW_LINE;
           end loop;
           PUT_LINE("Above are the procedures/functions and root faults.");
           PUT_LINE("+++++++++++++++++++++++++++++++++++++++++++++++");
          else
           PUT_LINE("There are no procedures defined within the input
                                code. ");
           NEW_LINE(5);
          end if;

          -- Code to print out the exception table of exceptions
          NEW_LINE(10);
          if EXCEPT_COUNTER - 1 > 0 then
           PUT    ("The number of exceptions on the table are ");
```

```
            PUT(EXCEPT_COUNTER - 1, width => 5);
            NEW_LINE(2);
            PUT_LINE("The exceptions and their root falults on the table ");
            PUT_LINE("are the following:  ");
            PUT_LINE("-------------------------------------------------------");
            NEW_LINE;
            for INDEX in 1..EXCEPT_COUNTER - 1 loop
                PUT_LINE(EX_TAB(INDEX).EXCCEPT_NAME);
                PUT_LINE(RETURN_ROOT_FAULT
                (EX_TAB(INDEX).EXCEPT_PTR));
                NEW_LINE;
            end loop;
            PUT_LINE("Above are the exceptions and their root faults.");
          else
                PUT_LINE("There are no defined exceptions within the input
                code. ");
                NEW_LINE(5);
          end if; } ;

compilation_unit :
        context_cl library_unit {$$ := $2;
                        -- Procedure to print nodes to screen
                        NEW_PAGE;
                        TREE_TRAVERSAL($2.NODE_T);

                        -- Procedure to create file for FTE interface
                        TEXT_IO.CREATE(FTE_FILE,
                                mode => OUT_FILE,
                                name => "NEW_FTE");
                        CREATE_FILE($2.NODE_T, FTE_FILE);
                        TEXT_IO.CLOSE(FTE_FILE);

                        -- Procedure to print text output for fte file
                        NEW_PAGE;
                        FTE_FILE1($2.NODE_T); }
    | context_cl secondary_unit ;

library_unit :
        subprg_d
    |   pkg_d
    |   gen_d
    |   gen_inst
    |   subprg_body {$$ := $1;}   ;

secondary_unit:
        library_unit_body | subunit;

library_unit_body :
        pkg_body_or_subprg_body ;

context_cl    : ..with_cl..use_cl.... ;
```

```
with_cl : WITH_TOKEN sim_n ...sim_n.. ';' ;

body_stub :
     subprg_spec IS_TOKEN SEPARATE_TOKEN ';'
  |  PACKAGE_TOKEN BODY_TOKEN sim_n IS_TOKEN SEPARATE_TOKEN ';'
  |  TASK_TOKEN BODY_TOKEN sim_n   IS_TOKEN SEPARATE_TOKEN ';'  ;

subunit : SEPARATE_TOKEN '(' expanded_n ')' proper_body ;

excptn_d : idents ':' EXCEPTION_TOKEN ';' ;

excptn_handler:
       WHEN_TOKEN excptn_choice ..or_excptn_choice.. ARROW
         seq_of_stmts
{$$ := (form   => nde, node_t  => BUILD_EXCEPTION
                                  ($2.NODE_T, $5.NODE_T));

     EX_TAB(EXCEPT_COUNTER).EXCEPT_NAME :=
               RETURN_ROOT_FAULT($2.NODE_T);
     EX_TAB(EXCEPT_COUNTER).EXCEPT_PTR  := $5.NODE_T;
     EXCEPT_COUNTER                     := EXCEPT_COUNTER + 1;  } ;

excptn_choice : expanded_n   {$$ := $1; }
        | OTHERS_TOKEN{$$ := (form   => nde,
                        node_t => NAME_TEMPLATE(YYTEXT)); } ;

raise_stmt : RAISE_TOKEN .expanded_n. ';'
         {$$ := (form   => nde,
             node_t => BUILD_RAISE($2.NODE_T)); } ;

gen_d : gen_spec ';' ;

gen_spec :
       gen_fml_part subprg_spec
  |    gen_fml_part pkg_spec ;

gen_fml_part :GENERIC_TOKEN ..gen_prm_d.. ;

gen_prm_d :
       idents ':' .IN.OUT.. ty_mk ._ASN_expr. ';'
  |  TYPE_TOKEN IDENTIFIER IS_TOKEN gen_ty_def ';'
  |    priv_ty_d
  |  WITH_TOKEN subprg_spec .IS_BOX_. ';' ;

gen_ty_def :
    '(' BOX ')' | RANGE_TOKEN BOX | DIGITS_TOKEN BOX | DELTA_TOKEN BOX
  |    array_ty_def     | access_ty_def  ;

gen_inst :
       PACKAGE_TOKEN IDENTIFIER IS_TOKEN
```

```
        NEW_TOKEN expanded_n .gen_act_part. ';'
  |     PROCEDURE__ident__IS_
        NEW_TOKEN expanded_n .gen_act_part. ';'
  |     FUNCTION_TOKEN  designator IS_TOKEN
          NEW_TOKEN expanded_n .gen_act_part. ';' ;

gen_act_part :
     '(' gen_asc ...gen_asc.. ')'          ;

gen_asc    :
       .gen_fml_prmARROW.gen_act_prm ;

gen_fml_prm :
       sim_n | op_symbol ;

gen_act_prm :
 expr_or_name_or_subprg_n_or_ent_n_or_ty_mk ;

rep_cl :
       ty_rep_cl | address_cl ;

ty_rep_cl : length_cl
  |     enum_rep_cl
  |     rec_rep_cl   ;

length_cl : FOR_TOKEN attribute USE_TOKEN sim_expr ';' ;

enum_rep_cl :
     FOR__ty_sim_n__USE_ aggr ';' ;

rec_rep_cl:
       FOR__ty_sim_n__USE_
         RECORD_TOKEN .algt_cl.
               ..cmpon_cl..
       END_TOKEN RECORD_TOKEN ';'  ;

algt_cl :    AT_TOKEN MOD_TOKEN sim_expr ';' ;

cmpon_cl :
     name AT_TOKEN sim_expr RANGE_TOKEN rng ';' ;

address_cl : FOR_TOKEN sim_n USE_TOKEN AT_TOKEN sim_expr ';' ;

code_stmt : ty_mk_rec_aggr ';' ;

..prag.. : {$$ := (form   => nde,
             node_t => null);}
  |     ..prag.. prag ;

.arg_ascs :
  |  '(' arg_ascs ')' ;
```

```
arg_ascs :
        arg_asc
  |  arg_ascs ',' arg_asc ;

._ASN_expr.:
  |     ASSIGNMENT expr ;

...ident.. :
  |  ...ident.. ',' IDENTIFIER  ;

.constrt. :
  |     constrt  ;

expanded_n :
        IDENTIFIER    {$$ := (form   => nde,
                      node_t => NAME_TEMPLATE(YYTEXT)); }
  |  expanded_n '.' IDENTIFIER  ;

...enum_lit_spec.. :
  |  ...enum_lit_spec.. ','
                  enum_lit_spec ;

.mg_c. :
  |     mg_c    ;

...idx_subty_def..      :
  |  ...idx_subty_def.. ',' idx_subty_def           ;

...dscr_mg.. :
  |  ...dscr_mg.. ',' dscr_mg  ;

..cmpon_d.. :
  |  ..cmpon_d.. cmpon_d ..prag..   ;

...discr_spec..:
  |  ...discr_spec.. ';' discr_spec  ;

..variant.. :
  |     ..variant.. variant ;

..or_choice.. :                {$$ := (form   => nde,
                               node_t => null ); }
  |  ..or_choice.. '|' choice   {$$ := (form   => nde,
                               node_t => BUILD_OR_CHOICE ($1.node_t,
                                              $3.node_t)); } ;

..basic_decl_item..   :
        ..prag..
  |     ..basic_decl_item.. basic_decl_item ..prag.. ;
```

```
..later_decl_item..    :
        ..prag..
  |  ..later_decl_item.. later_decl_item  prag.. ;

...cmpon_asc..   :
  |  ...cmpon_asc.. ',' cmpon_asc ;

rel..AND__rel.. :
        rel AND_TOKEN rel          {$$ := (form   => nde,
                                node_t => BUILD_REL_AND ($1.node_t,
                                          $3.node_t));}
  |     rel..AND__rel.. AND_TOKEN rel  {$$ := (form   => nde,
                                node_t => BUILD_REL_AND ($1.node_t,
                                          $3.node_t));} ;

rel..OR__rel.. :
        rel OR_TOKEN rel           {$$ := (form   => nde,
                                node_t => BUILD_REL_OR ($1.node_t,
                                          $3.node_t));}
  |     rel..OR__rel.. OR_TOKEN rel   {$$ := (form   => nde,
                                node_t => BUILD_REL_OR ($1.node_t,
                                          $3.node_t));};

rel..XOR__rel.. :
        rel {$$ := $1;}
  |     ..XOR__rel.. ;

..XOR__rel.. :
        rel XOR_TOKEN rel
  |     ..XOR__rel.. XOR_TOKEN rel;

rel..AND__THEN__rel.. ;
        rel AND_TOKEN THEN_TOKEN rel   {$$ := (form   => nde,
                                node_t => BUILD_REL_AND_THEN ($1.node_t,
                                          $4.node_t));}
  |     rel..AND__THEN__rel.. AND_TOKEN THEN_TOKEN rel
                          {$$ := (form   => nde,
                              node_t => BUILD_REL_AND_THEN ($1.node_t,
                                          $4.node_t));};

rel..OR__ELSE__rel..:
        rel OR_TOKEN ELSE_TOKEN rel   {$$ := (form   => nde,
                                node_t => BUILD_REL_OR_ELSE ($1.node_t,
                                          $4.node_t));}
  |     rel..OR__ELSE__rel.. OR_TOKEN ELSE_TOKEN rel
                          {$$ := (form   => nde,
                              node_t => BUILD_REL_OR_ELSE ($1.node_t,
                                          $4.node_t));};

relal_op__sim_expr. :   {$$ := (form   => nde,
                        node_t => null);}
```

```
  |    relal_op sim_expr  {$$ := $2;} ;

sim_expr.NOT.IN__rng_or_sim_expr.NOT.IN__ty_mk:
       sim_expr .NOT. IN_TOKEN rng ;

.NOT. :
  |    NOT_TOKEN   ;

.unary_add_op.term..binary_add_op__term.. :
       term {$$ := $1;}
  |    unary_add_op term
  |    .unary_add_op.term..binary_add_op__term..
       binary_add_op term {$$ := (form  => nde,
                    node_t => ADD_SUB_TEMPLATE ($1.node_t,
                                          $2.node_t,
                                          $3.node_t));} ;

factor..mult_op__factor..:
       factor {$$ := $1;}
  |    factor..mult_op__factor.. mult_op factor {$$ := (form  => nde,
                                    node_t => DIV_MULT_TEMPLATE
                                          ($1.NODE_T,
                                          $3.NODE_T,
                                          $2.NODE_T));} ;

._EXP___pri. :
  |    DOUBLE_STAR pri ;

ty_mkaggr_or_ty_mkPexprP_ :
     prefix ' aggr ;

..stmt.. :
       ..prag.. {$$ := $1;}

  |    ..stmt.. stmt ..prag.. {$$ := (form  => nde,
                        node_t => STATEMENTS ($1.NODE_T,
                                          $2.NODE_T));};


..label.. :
  |    ..label.. label ;

..ELSIF__cond__THEN__seq_of_stmts.. : {$$ := (form  => nde,
                              node_t => null);}
  |    ..ELSIF__cond__THEN__seq_of_stmts..
               ELSIF_TOKEN cond THEN_TOKEN
                 seq_of_stmts     {$$ := (form  => nde,
                              node_t => ELSIF_TEMPLATE($3.node_t,
                                          $5.node_t,
                                          $1.node_t));};
```

```
.ELSE__seq_of_stmts.:  {$$ := (form  => nde,
                        node_t => null);}
 |    ELSE_TOKEN
        seq_of_stmts   {$$ := (form  => nde,
                        node_t => ELSE_TEMPLATE($2.node_t)); };

case_stmt_alt..case_stmt_alt.. :
        ..prag..
        case_stmt_alt
        ..case_stmt_alt..
       {$$ := (form  => nde,
           node_t => BUILD_CASE_ALT_1_MORE($2.node_t,
                                $3.node_t)); } ;

..case_stmt_alt..:          {$$ := (form  => nde,
                                node_t => null); }
 |    ..case_stmt_alt.. case_stmt_alt {$$ := (form  => nde,
                                node_t => BUILD_CASE_ALT_0_MORE
                                      ($1.node_t,
                                       $2.node_t)); } ;

.sim_nC.:   {$$ := (form  => nde,
                    node_t => null); }
 |  sim_n ':'  {$$ := $1; } ;

.sim_n. :
 |    sim_n ;

.iteration_scheme. :    {$$ := (form  => nde,
                        node_t => null); }
 |    iteration_scheme  {$$ := $1; } ;

.REVERSE. :             {$$ := (form  => nde,
                        node_t => null); }
 |    REVERSE_TOKEN     {$$ := (form  => nde,
                        node_t => BUILD_REVERSE_NODE); } ;

.DECLARE__decl_part.   :
 |    DECLARE_TOKEN
        decl_part       ;

.EXCEPTION__excptn_handler..excptn_handler... : {$$ := (form  => nde,
                                node_t => null); }
 |    EXCEPTION_TOKEN
        ..prag.. excptn_handlers      {$$ := $3; } ;

excptn_handlers:
       excptn_handler                 {$$ := $1; }
 |     excptn_handlers excptn_handler       {$$ := (form  => nde,
                                node_t => JOIN_EXCEPTIONS($1.NODE_T,
                                        $2.NODE_T)); } ;
```

```
.expanded_n. :    {$$ := (form  => nde,
                  node_t => null); }
  |    expanded_n  {$$ := $1;} ;

.WHEN__cond. :
  |    WHEN_TOKEN cond;

.expr. :
  |    expr ;

.fml_part. :
  |    fml_part ;

..._prm_spec.. :
  |  ..._prm_spec.. ';' prm_spec ;

.IN. ;
  |    IN_TOKEN ;

.decl_part.  : decl_part;

.designator. :
  |  designator  ;

.PRIVATE..basic_decl_item... :
  |    PRIVATE_TOKEN
         ..basic_decl_item.. ;

.BEGIN__seq_of_stmts.EXCEPTION__excptn_handler..excptn_handler...
                                  ;
  |    BEGIN_TOKEN
         seq_of_stmts
       .EXCEPTION__excptn_handler..excptn_handler...
    ;

.LIMITED. :
  |    LIMITED_TOKEN  ;

...expanded_n.. :
  |  ...expanded_n.. ',' expanded_n ;

.TYPE.:
  |    TYPE_TOKEN ;

.IS..ent_d_..rep_cl.END.sim_n. :
  |    IS_TOKEN
         ..ent_d..
         ..rep_cl..
       END_TOKEN .sim_n.   ;
```

```
..ent_d.. :
        ..prag..
  |     ..ent_d.. ent_d ..prag..;

..rep_cl.. :
  |     ..rep_cl.. rep_cl ..prag..;

.Pent_idx_P..fml_part. :
        .fml_part.
  |  '(' ent_idx ')' .fml_part. ;

.DO__seq_of_stmts__END.sim_n.. :
  |     DO_TOKEN
          seq_of_stmts
        END_TOKEN .sim_n. ;

..OR__select_alt.. :
  |     ..OR__select_alt.. OR_TOKEN select_alt;

.WHEN__condARROW.selec_wait_alt :
        selec_wait_alt
  |     WHEN_TOKEN cond ARROW selec_wait_alt ;

accept_stmt.seq_of_stmts. :
        ..prag.. accept_stmt .seq_of_stmts. ;

delay_stmt.seq_of_stmts. :
        ..prag.. delay_stmt .seq_of_stmts. ;

TERM_stmt  : ..prag.. TERMINATE_TOKEN ';' ..prag.. ;

.seq_of_stmts.:
        ..prag..
  |     seq_of_stmts;

...name..:
  |  ...name.. '.' name ;

..compilation_unit.. :
        ..prag..
  |     ..compilation_unit.. compilation_unit ..prag.. ;

pkg_body_or_subprg_body  : pkg_body ;

..with_cl..use_cl.... :
  |     ..with_cl..use_cl.... with_cl use_cls ;

use_cls :
        ..prag..
  |     use_cls use_cl ..prag.. ;
```

```
...sim_n.. :
  |  ...sim_n.. ',' sim_n ;

..or_excptn_choice.. :
  |  ..or_excptn_choice.. '|' excptn_choice ;

..gen_prm_d.. :
  |     ..gen_prm_d.. gen_prm_d ;

.IN.OUT.. :
        .IN.
  |     IN_TOKEN OUT_TOKEN ;

.IS_BOX_.:
  |     IS_TOKEN name
  |     IS_TOKEN BOX ;

PROCEDURE__ident__IS_  : subprg_spec IS_TOKEN ;

.gen_act_part. :
  |     gen_act_part ;

...gen_asc.. :
  |  ...gen_asc.. ',' gen_asc ;

.gen_fml_prmARROW.gen_act_prm :
        gen_act_prm
  |     gen_fml_prm ARROW gen_act_prm ;

expr_or_name_or_subprg_n_or_ent_n_or_ty_mk
                                    : expr ;

FOR__ty_sim_n__USE_ :
        FOR_TOKEN sim_n USE_TOKEN;


.algt_cl. :
        ..prag..
  |  ..prag.. algt_cl ..prag..  ;

..cmpon_cl.. :
  |  ..cmpon_cl.. cmpon_cl ..prag.. ;

ty_mk_rec_aggr : qualified_expr    ;

%%

with fault_tree_generator, trace_package, traverse_pkg;
use fault_tree_generator, trace_package, traverse_pkg;

package parser is
```

```
   procedure yyparse;

   echo           : boolean := false;
   number_of_errors : natural := 0;

end parser;

with ada_tokens, ada_goto, ada_shift_reduce, ada_lex, ada_lex_dfa, text_io;
use  ada_tokens, ada_goto, ada_shift_reduce, ada_lex, ada_lex_dfa, text_io;
package body parser is

   package INTEGER_INOUT is new INTEGER_IO(INTEGER);
   use    INTEGER_INOUT;

   procedure yyerror(s: in string := "syntax error") is
   begin
        number_of_errors := number_of_errors + 1;
        put("<<< *** ");
        put_line(s);
   end yyerror;


##%procedure_parse

end parser;
```

B. AFLEX SPECIFICATION FILE

```
--/*-------------------------------------------------------------------*/
--/* Lexical input for LEX for LALR(1) Grammar for ANSI Ada          */
--/*                                                                  */
--/*          Herman Fischer                                          */
--/*        Litton Data Systems                                       */
--/*          March 26, 1984                                          */
--/*                                                                  */
--/* Accompanies Public Domain YACC format Ada grammar               */
--/*-------------------------------------------------------------------*/

%START IDENT Z

A               [aA]
B               [bB]
C               [cC]
D               [dD]
E               [eE]
F               [fF]
G               [gG]
H               [hH]
I               [iI]
J               [jJ]
K               [kK]
L               [lL]
M               [mM]
N               [nN]
O               [oO]
P               [pP]
Q               [qQ]
R               [rR]
S               [sS]
T               [tT]
U               [uU]
V               [vV]
W               [wW]
X               [xX]
Y               [yY]
Z               [zZ]

%%
{A}{B}{O}{R}{T}{ECHO; ENTER(Z); return(ABORT_TOKEN);}
{A}{B}{S}       {ECHO; ENTER(Z); return(ABS_TOKEN);}
{A}{C}{C}{E}{P}{T}{ECHO; ENTER(Z); return(ACCEPT_TOKEN);}
{A}{C}{C}{E}{S}{S}{ECHO; ENTER(Z); return(ACCESS_TOKEN);}
{A}{L}{L}       {ECHO; ENTER(Z); return(ALL_TOKEN);}
{A}{N}{D}       {ECHO; ENTER(Z); return(AND_TOKEN);}
{A}{R}{R}{A}{Y}{ECHO; ENTER(Z); return(ARRAY_TOKEN);}
{A}{T}                  {ECHO; ENTER(Z); return(AT_TOKEN);}
{B}{E}{G}{I}{N}{ECHO; ENTER(Z); return(BEGIN_TOKEN);}
```

```
{B}{O}{D}{Y}{ECHO; ENTER(Z); return(BODY_TOKEN);}
{C}{A}{S}{E}{ECHO; ENTER(Z); return(CASE_TOKEN);}
{C}{O}{N}{S}{T}{A}{N}{T}{ECHO; ENTER(Z); return(CONSTANT_TOKEN);}
{D}{E}{C}{L}{A}{R}{E} {ECHO; ENTER(Z); return(DECLARE_TOKEN);}
{D}{E}{L}{A}{Y}    {ECHO; ENTER(Z); return(DELAY_TOKEN);}
{D}{E}{L}{T}{A}    {ECHO; ENTER(Z); return(DELTA_TOKEN);}
{D}{I}{G}{I}{T}{S}   {ECHO; ENTER(Z); return(DIGITS_TOKEN);}
{D}{O}         {ECHO; ENTER(Z); return(DO_TOKEN);}
{E}{L}{S}{E}      {ECHO; ENTER(Z); return(ELSE_TOKEN);}
{E}{L}{S}{I}{F}    {ECHO; ENTER(Z); return(ELSIF_TOKEN);}
{E}{N}{D}        {ECHO; ENTER(Z); return(END_TOKEN);}
{E}{N}{T}{R}{Y}    {ECHO; ENTER(Z); return(ENTRY_TOKEN);}
{E}{X}{C}{E}{P}{T}{I}{O}{N} /ECHO; ENTER(Z); return(EXCEPTION_TOKEN);}
{E}{X}{I}{T}      {ECHO; ENTER(Z); return(EXIT_TOKEN);}
{F}{O}{R}        {ECHO; ENTER(Z); return(FOR_TOKEN);}
{F}{U}{N}{C}{T}{I}{O}{N} {ECHO; ENTER(Z); return(FUNCTION_TOKEN);}
{G}{E}{N}{E}{R}{I}{C} {ECHO; ENTER(Z); return(GENERIC_TOKEN);}
{G}{O}{T}{O}      {ECHO; ENTER(Z); return(GOTO_TOKEN);}
{I}{F}         {ECHO; ENTER(Z); return(IF_TOKEN);}
{I}{N}         {ECHO; ENTER(Z); return(IN_TOKEN);}
{I}{S}         {ECHO; ENTER(Z); return(IS_TOKEN);}
{L}{I}{M}{I}{T}{E}{D} {ECHO; ENTER(Z); return(LIMITED_TOKEN);}
{L}{O}{O}{P}      {ECHO; ENTER(Z); return(LOOP_TOKEN);}
{M}{O}{D}        {ECHO; ENTER(Z); return(MOD_TOKEN);}
{N}{E}{W}        {ECHO; ENTER(Z); return(NEW_TOKEN);}
{N}{O}{T}        {ECHO; ENTER(Z); return(NOT_TOKEN);}
{N}{U}{L}{L}      {ECHO; ENTER(Z); return(NULL_TOKEN);}
{O}{F}         {ECHO; ENTER(Z); return(OF_TOKEN);}
{O}{R}         {ECHO; ENTER(Z); return(OR_TOKEN);}
{O}{T}{H}{E}{R}{S}   {ECHO; ENTER(Z); return(OTHERS_TOKEN);}
{O}{U}{T}        {ECHO; ENTER(Z); return(OUT_TOKEN);}
{P}{A}{C}{K}{A}{G}{E} {ECHO; ENTER(Z); return(PACKAGE_TOKEN);}
{P}{R}{A}{G}{M}{A}   {ECHO; ENTER(Z); return(PRAGMA_TOKEN);}
{P}{R}{I}{V}{A}{T}{E} {ECHO; ENTER(Z); return(PRIVATE_TOKEN);}
{P}{R}{O}{C}{E}{D}{U}{R}{E} {ECHO; ENTER(Z); return(PROCEDURE_TOKEN);}
{R}{A}{I}{S}{E}    {ECHO; ENTER(Z); return(RAISE_TOKEN);}
{R}{A}{N}{G}{E}    {ECHO; ENTER(Z); return(RANGE_TOKEN);}
{R}{E}{C}{O}{R}{D}   {ECHO; ENTER(Z); return(RECORD_TOKEN);}
{R}{E}{M}      {ECHO; ENTER(Z); return(REM_TOKEN);}
{R}{E}{N}{A}{M}{E}{S}{ECHO; ENTER(Z); return(RENAMES_TOKEN);}
{R}{E}{T}{U}{R}{N}{ECHO; ENTER(Z); return(RETURN_TOKEN);}
{R}{E}{V}{E}{R}{S}{E}{ECHO; ENTER(Z); return(REVERSE_TOKEN);}
{S}{E}{L}{E}{C}{T}{ECHO; ENTER(Z); return(SELECT_TOKEN);}
{S}{E}{P}{A}{R}{A}{T}{E}{ECHO; ENTER(Z); return(SEPARATE_TOKEN);}
{S}{U}{B}{T}{Y}{P}{E}{ECHO; ENTER(Z); return(SUBTYPE_TOKEN);}
{T}{A}{S}{K}{ECHO; ENTER(Z); return(TASK_TOKEN);}
{T}{E}{R}{M}{I}{N}{A}{T}{E}{ECHO; ENTER(Z); return(TERMINATE_TOKEN);}
{T}{H}{E}{N}{ECHO; ENTER(Z); return(THEN_TOKEN);}
{T}{Y}{P}{E}{ECHO; ENTER(Z); return(TYPE_TOKEN);}
{U}{S}{E}      {ECHO; ENTER(Z); return(USE_TOKEN);}
{W}{H}{E}{N}{ECHO; ENTER(Z); return(WHEN_TOKEN);}
```

```
{W}{H}{I}{L}{E}{ECHO; ENTER(Z); return(WHILE_TOKEN);}
{W}{I}{T}{H}{ECHO; ENTER(Z); return(WITH_TOKEN);}
{X}{O}{R}     {ECHO; ENTER(Z); return(XOR_TOKEN);}
"=>"    {ECHO; ENTER(Z); return(ARROW);}
".."    {ECHO; ENTER(Z); return(DOUBLE_DOT);}
"**"    {ECHO; ENTER(Z); return(DOUBLE_STAR);}
":="    {ECHO; ENTER(Z); return(ASSIGNMENT);}
"/="    {ECHO; ENTER(Z); return(INEQUALITY);}
">="    {ECHO; ENTER(Z); return(GREATER_THAN_OR_EQUAL);}
"<="    {ECHO; ENTER(Z); return(LESS_THAN_OR_EQUAL);}
"<<"    {ECHO; ENTER(Z); return(LEFT_LABEL_BRACKET);}
">>"    {ECHO; ENTER(Z); return(RIGHT_LABEL_BRACKET);}
"<>"    {ECHO; ENTER(Z); return(BOX);}
"&"     {ECHO; ENTER(Z); return('&'); }
"("     {ECHO; ENTER(Z); return('('); }
")"     {ECHO; ENTER(IDENT); return(')'); }
"*"     {ECHO; ENTER(Z); return('*'); }
"+"     {ECHO; ENTER(Z); return('+'); }
","     {ECHO; ENTER(Z); return(','); }
"-"     {ECHO; ENTER(Z); return('-'); }
"."     {ECHO; ENTER(Z); return('.'); }
"/"     {ECHO; ENTER(Z); return('/'); }
":"     {ECHO; ENTER(Z); return(':'); }
";"     {ECHO; ENTER(Z); return(';'); }
"<"     {ECHO; ENTER(Z); return('<'); }
"="     {ECHO; ENTER(Z); return('='); }
">"     {ECHO; ENTER(Z); return(>'); }
"|"     {ECHO; ENTER(Z); return('|'); }
<IDENT>\'  {ECHO; ENTER(Z); return('');}

[a-z_A-Z][a-z_A-Z0-9]*      {ECHO; ENTER(IDENT);return(IDENTIFIER);}
[0-9][0-9_]*([.][0-9_]+)?([Ee][-+]?[0-9_]+)? {
                                ECHO; ENTER(Z);
                                return(INTEGER_LITERAL);}

[0-9][0-9_]*#[0-9a-fA-F_]+([.][0-9a-fA-F_]+)?#([Ee][-+]?[0-9_]+)? {
                                ECHO; ENTER(Z);
                                return(INTEGER_LITERAL);}

\"([^"]*(\"\")*)*\" {ECHO; ENTER(Z); return(STRING_LITERAL);}

<Z>\'([^']|\'\')\'   {ECHO; ENTER(Z); return(CHARACTER_LITERAL);}

[\t]  ECHO;        -- ignore spaces and tabs
"--".* ECHO;        -- ignore comments to end-of-line

.          {ECHO;
           text_io.put_line("?? lexical error" & ada_lex_dfa.yytext & "??");
             num_errors := num_errors + 1;}
[\n]    {ECHO; linenum;}
%%
```

```
with ada_tokens;
use  ada_tokens;
use text_io;

package ada_lex is

 lines          : positive := 1;
 num_errors : natural := 0;
 procedure linenum;
 function yylex return token;

end ada_lex;

package body ada_lex is

 procedure linenum is
 begin
  text_io.put(integer'image(lines) & ":");
  lines := lines + 1;
 end linenum;

##

end ada_lex;
```

APPENDIX B: SOURCE CODE FOR TEMPLATE GENERATOR TOOL

A. PACKAGE SPECIFICTION FOR FAULT TREE GENERATOR

```
with TEXT_IO, TRACE_PACKAGE;
use TEXT_IO, TRACE_PACKAGE;

package FAULT_TREE_GENERATOR is

package INTEGER_INOUT is new INTEGER_IO(INTEGER);
use   INTEGER_INOUT;

package BOOLEAN_INOUT is new ENUMERATION_IO(BOOLEAN);
use   BOOLEAN_INOUT;

MAX_CHAR_LONG   : INTEGER  := 80;
MAX_CHAR_SHORT  : INTEGER  := 10;
MAX_ENTRY       : INTEGER  := 100;

subtype GATE_TYPE   is INTEGER range 0..2;
subtype NODE_TYPE   is INTEGER range 1..6;

-- Represents types of gates for fault tree
NO_GATE         : CONSTANT GATE_TYPE := 0;
AND_GATE        : CONSTANT GATE_TYPE := 1;
OR_GATE         : CONSTANT GATE_TYPE := 2;

-- Represents figures for fault tree
RECTANGLE       : CONSTANT NODE_TYPE := 1;
CIRCLE          : CONSTANT NODE_TYPE := 2;
DIAMOND         : CONSTANT NODE_TYPE := 3;
ELLIPSE         : CONSTANT NODE_TYPE := 4;
HOUSE           : CONSTANT NODE_TYPE := 5;
TRIANGLE        : CONSTANT NODE_TYPE := 6;

-- Data structure to represent a node within fault tree
type ST_LIST_NODE    is private;
type ST_LIST_NODE_PTR is access ST_LIST_NODE;

-- Data structure to represent output locations
type OUTPUT_LOCATION is private;
type OUTPUT_LOCATION_PTR is access OUTPUT_LOCATION;

-- Data structure for system table records
type TAB_ENTRY is
 record
  P_F_NAME     : STRING(1..MAX_CHAR_LONG) := (OTHERS => ' ');
  STMT_PTR     : ST_LIST_NODE_PTR;
```

```
 end record;

-- Data structure representing system table
type SYS_TABLE is ARRAY(1..MAX_ENTRY) of TAB_ENTRY;

-- Data structure for exception table records
type EXCEPT_ENTRY is
 record
  EXCEPT_NAME  : STRING(1..MAX_CHAR_LONG) := (OTHERS => ' ');
  EXCEPT_PTR   : ST_LIST_NODE_PTR;
 end record;

-- Data structure representing the exception table
type EXCEPT_TABLE is array (1..100) of EXCEPT_ENTRY;

-------------------------------------------------------------------------------
-- Functions to access part of the data structure, due to private type

function RETURN_CHILD (FIRST : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function RETURN_CHILD_PREV (FIRST : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function RETURN_ROOT_FAULT (FIRST : ST_LIST_NODE_PTR)
 return STRING;

function RETURN_GATE_TYPE (FIRST : ST_LIST_NODE_PTR)
 return GATE_TYPE;

function RETURN_NO_CHILDREN (FIRST : ST_LIST_NODE_PTR)
 return NATURAL;

function RETURN_NODE_TYPE (FIRST : ST_LIST_NODE_PTR)
 return NODE_TYPE;

function RETURN_X_COORD (FIRST : ST_LIST_NODE_PTR)
 return INTEGER;

function RETURN_Y_COORD (FIRST : ST_LIST_NODE_PTR)
 return INTEGER;

function RETURN_START_L (FIRST : ST_LIST_NODE_PTR)
 return NATURAL;

function RETURN_END_L (FIRST : ST_LIST_NODE_PTR)
 return NATURAL;

function RETURN_FILE_NAME (FIRST : ST_LIST_NODE_PTR)
 return STRING;
```

```
function RETURN_LABEL (FIRST : ST_LIST_NODE_PTR)
 return STRING;

-----------------------------------------------------------------------------

-- Procedures to put values onto node structure

procedure PUSH_LABEL (FIRST  : ST_LIST_NODE_PTR;
                      LABEL  : STRING);

procedure PUSH_FILE_N (FIRST  : ST_LIST_NODE_PTR;
                       LABEL  : STRING);

procedure PUSH_X_COORD (FIRST  : ST_LIST_NODE_PTR;
                        NUMBER : INTEGER);

procedure PUSH_Y_COORD (FIRST  : ST_LIST_NODE_PTR;
                        NUMBER : INTEGER);

-----------------------------------------------------------------------------

-- Procedures to print the values of the nodes

procedure PRINT_NODE (NODE : in ST_LIST_NODE_PTR);

procedure PRINT_CHILDREN_NODES (NODE : in  ST_LIST_NODE_PTR);

procedure PRINT_CHILDREN_OF_CHILDREN (NODE : in  out ST_LIST_NODE_PTR);

procedure PRINT_CHILDREN_OF_CHILDREN_PREVIOUS (NODE : in out
ST_LIST_NODE_PTR);

-----------------------------------------------------------------------------

-- Functions to build templates

function NAME_TEMPLATE (ID : STRING) return ST_LIST_NODE_PTR;

function VALUE_TEMPLATE (ID : STRING) return ST_LIST_NODE_PTR;

function DIVISION_BY_ZERO_TEMPLATE return ST_LIST_NODE_PTR;

function ADD_FAULT_TEMPLATE return ST_LIST_NODE_PTR;

function SUB_FAULT_TEMPLATE return ST_LIST_NODE_PTR;

function ADD_SUB_TEMPLATE (FIRST, SECOND, THIRD : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function DIV_MULT_TEMPLATE (FIRST, SECOND, THIRD : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function INDEX_COMP_TEMPLATE (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;
```

```
function ASSIGNMENT_TEMPLATE (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function IF_STATEMENT_TEMPLATE (FIRST, SECOND, THIRD, FOURTH :
ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function ELSIF_TEMPLATE (FIRST, SECOND, THIRD : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function ELSE_TEMPLATE (FIRST : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function STATEMENTS (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function BUILD_REL (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function BUILD_REL_AND (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function BUILD_REL_OR (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function BUILD_REL_AND_THEN (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function BUILD_REL_OR_ELSE (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function BUILD_RANGE_NAME (FIRST : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function BUILD_RNG (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function BUILD_DESCR_RNG_1 (FIRST : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function BUILD_DESCR_RNG_2 (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function BUILD_REVERSE_NODE
 return ST_LIST_NODE_PTR;

function BUILD_LOOP_PRM_SPEC (FIRST, SECOND, THIRD : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function BUILD_FOR_ITERATION (FIRST : ST_LIST_NODE_PTR)
```

```
 return ST_LIST_NODE_PTR;

function BUILD_WHILE_ITERATION (FIRST : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function BUILD_LOOP_STMT (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function BUILD_CASE_STMT_TEMPLATE (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function BUILD_CASE_ALT_1_MORE (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function BUILD_CASE_ALT_0_MORE (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function BUILD_CASE_STMT_ALT (FIRST,
                  SECOND,
                  THIRD     : ST_LIST_NODE_PTR;
                  OTHERS_CHK : BOOLEAN)
 return ST_LIST_NODE_PTR;

function BUILD_OR_CHOICE (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function BUILD_CHOICE_2 (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function BUILD_CHOICE_3 (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function BUILD_PROC_CALL (FIRST   : ST_LIST_NODE_PTR;
                TABLE   : SYS_TABLE;
                COUNTER : INTEGER)
 return ST_LIST_NODE_PTR;

function BUILD_BLOCK_STMT (FIRST   : ST_LIST_NODE_PTR;
                SECOND  : ST_LIST_NODE_PTR;
                THIRD   : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function JOIN_EXCEPTIONS (FIRST   : ST_LIST_NODE_PTR;
                SECOND  : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function BUILD_EXCEPTION (FIRST   : ST_LIST_NODE_PTR;
                SECOND  : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function CHECK_FUNCTION (FIRST   : ST_LIST_NODE_PTR;
```

```
                TABLE   : SYS_TABLE;
                COUNTER : INTEGER)
 return ST_LIST_NODE_PTR;

function BUILD_RAISE (FIRST : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR;

function CHECK_EXCEPT (FIRST       : ST_LIST_NODE_PTR;
                EX_TAB      : EXCEPT_TABLE;
                EXCEPT_COUNTER : INTEGER)
 return ST_LIST_NODE_PTR;

-------------------------------------------------------------------------------

private

type ST_LIST_NODE is
   record
      ST_NODE_LABEL       : STRING(1..MAX_CHAR_SHORT) := (others => ' ');
      ROOT_FAULT          : STRING(1..MAX_CHAR_LONG)  := (others => ' ');
      FILE_NAME           : STRING(1..MAX_CHAR_LONG)  := (others => ' ');
      START_LINE          : NATURAL                   := 0;
      END_LINE            : NATURAL                   := 0;
      X_COORDINATE        : INTEGER                   := 0;
      Y_COORDINATE        : INTEGER                   := 0;
      TYPE_NODE           : NODE_TYPE                 := 1;
      TYPE_GATE           : GATE_TYPE                 := 0;
      NUMBER_OF_CHILDREN  : NATURAL                   := 0;
      PREV_ST_PTR         : ST_LIST_NODE_PTR          := NULL;
      CHILDREN_NODES      : ST_LIST_NODE_PTR          := NULL;
      OUTPUT_FLAG         : BOOLEAN                   := FALSE;
   end record;

type OUTPUT_LOCATION is
   record
      OUTPUT_LABEL        : STRING(1..MAX_CHAR_LONG)  := (others => ' ');
      OUTPUT_LINE         : NATURAL                   := 0;
      OUTPUT_ST_START_NODE : ST_LIST_NODE_PTR         := NULL;
      PREV_OUTPUT_LOCATION : OUTPUT_LOCATION_PTR      := NULL;
      NEXT_OUTPUT_LOCATION : OUTPUT_LOCATION_PTR      := NULL;
   end record;

end FAULT_TREE_GENERATOR;

-------------------------------------------------------------------------------
```

B. PACKAGE BODY FOR FAULT TREE GENERATOR

```
package body FAULT_TREE_GENERATOR is

-----------------------------------------------------------------------------
-- Function to return child to other package for visibility

function RETURN_CHILD (FIRST : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP : ST_LIST_NODE_PTR;

begin -- RETURN CHILD

 TEMP := FIRST.CHILDREN_NODES;
 RETURN TEMP;

end RETURN_CHILD;

-----------------------------------------------------------------------------
-- Function to return child prev to other package for visibility

function RETURN_CHILD_PREV (FIRST : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP : ST_LIST_NODE_PTR;

begin -- RETURN CHILD PREV

 TEMP := FIRST.PREV_ST_PTR;
 RETURN TEMP;

end RETURN_CHILD_PREV;

-----------------------------------------------------------------------------
-- Function to return root fault to other package for visibility

function RETURN_ROOT_FAULT (FIRST : ST_LIST_NODE_PTR)
 return STRING is

begin -- RETURN ROOT FAULT

 RETURN FIRST.ROOT_FAULT;

end RETURN_ROOT_FAULT;

-----------------------------------------------------------------------------
-- Function to return gate type to other package for visibility

function RETURN_GATE_TYPE (FIRST : ST_LIST_NODE_PTR)
 return GATE_TYPE is
```

```
begin -- RETURN GATE TYPE

 RETURN FIRST.TYPE_GATE;

end RETURN_GATE_TYPE;

-------------------------------------------------------------------------------
-- Function to return number of children to other package for visibility

function RETURN_NO_CHILDREN (FIRST : ST_LIST_NODE_PTR)
 return NATURAL is

begin -- RETURN NO CHILDREN

 RETURN FIRST.NUMBER_OF_CHILDREN;

end RETURN_NO_CHILDREN;

-------------------------------------------------------------------------------
-- Function to return node type to other package for visibility

function RETURN_NODE_TYPE (FIRST : ST_LIST_NODE_PTR)
 return NODE_TYPE is

begin -- RETURN NODE_TYPE

 RETURN FIRST.TYPE_NODE;

end RETURN_NODE_TYPE;

-------------------------------------------------------------------------------
-- Function to return x coordinate to other package for visibility

function RETURN_X_COORD (FIRST : ST_LIST_NODE_PTR)
 return INTEGER is

begin -- RETURN X_COORDINATE

 RETURN FIRST.X_COORDINATE;

end RETURN_X_COORD;

-------------------------------------------------------------------------------
-- Function to return y coordinate to other package for visibility

function RETURN_Y_COORD (FIRST : ST_LIST_NODE_PTR)
 return INTEGER is

begin -- RETURN Y COORDINATE
```

```
 RETURN FIRST.Y_COORDINATE;

end RETURN_Y_COORD;


-----------------------------------------------------------------------------
-- Function to return node type to other package for visibility

function RETURN_START_L (FIRST : ST_LIST_NODE_PTR)
 return NATURAL is

begin -- RETURN START L

 RETURN FIRST.START_LINE;

end RETURN_START_L;

-----------------------------------------------------------------------------
-- Function to return end line to other package for visibility

function RETURN_END_L (FIRST : ST_LIST_NODE_PTR)
 return NATURAL is

begin -- RETURN END L

 RETURN FIRST.END_LINE;

end RETURN_END_L;

-----------------------------------------------------------------------------
-- Function to return file name to other package for visibility

function RETURN_FILE_NAME (FIRST : ST_LIST_NODE_PTR)
 return STRING is

begin -- RETURN FILE NAME

 RETURN FIRST.FILE_NAME;

end RETURN_FILE_NAME;

-----------------------------------------------------------------------------
-- Function to return label to other package for visibility

function RETURN_LABEL (FIRST : ST_LIST_NODE_PTR)
 return STRING is

begin -- RETURN LABEL

 RETURN FIRST.ST_NODE_LABEL;
```

```
end RETURN_LABEL;

-----------------------------------------------------------------------------
-- Procedure to push label into data structure
procedure PUSH_LABEL (FIRST  : ST_LIST_NODE_PTR;
                      LABEL  : STRING) is

begin

 FIRST.ST_NODE_LABEL(LABEL'range) := LABEL;

end PUSH_LABEL;

-----------------------------------------------------------------------------
-- Procedure to push file name into data structure
procedure PUSH_FILE_N (FIRST  : ST_LIST_NODE_PTR;
                       LABEL  : STRING) is

begin

 FIRST.FILE_NAME(LABEL'range) := LABEL;

end PUSH_FILE_N;

-----------------------------------------------------------------------------
-- Procedure to push x coordinate into data structure
procedure PUSH_X_COORD (FIRST  : ST_LIST_NODE_PTR;
                        NUMBER : INTEGER) is

begin

 FIRST.X_COORDINATE := NUMBER;

end PUSH_X_COORD;

-----------------------------------------------------------------------------
-- Procedure to push y coordinate into data structure
procedure PUSH_Y_COORD (FIRST  : ST_LIST_NODE_PTR;
                        NUMBER : INTEGER) is

begin

 FIRST.Y_COORDINATE := NUMBER;

end PUSH_Y_COORD;

-----------------------------------------------------------------------------
-- Procedure to print the values of a node

procedure PRINT_NODE (NODE : in ST_LIST_NODE_PTR) is
```

```
begin -- PRINT NODE

 if NODE /= NULL then
  new_line (2);
  put_line ("Printing values of current node. ");
  put    ("ST_NODE_LABEL    : ");
  put_line (NODE.ST_NODE_LABEL);
  put    ("ROOT_FAULT       : ");
  put_line (NODE.ROOT_FAULT);
  put    ("FILE_NAME        : ");
  put_line (NODE.FILE_NAME);
  put    ("START_LINE       : ");
  put    (NODE.START_LINE);
  new_line;
  put    ("END_LINE         : ");
  put    (NODE.END_LINE);
  new_line;
  put    ("X_COORDINATE     : ");
  put    (NODE.X_COORDINATE);
  new_line;
  put    ("Y-COORDINATE     : ");
  put    (NODE.Y_COORDINATE);
  new_line;
  put    ("TYPE_NODE        : ");
  put    (NODE.TYPE_NODE);
  new_line;
  put    ("TYPE_GATE        : ");
  put    (NODE.TYPE_GATE);
  new_line;
  put    ("NUMBER_OF_CHILDREN: ");
  put    (NODE.NUMBER_OF_CHILDREN);
  new_line;
  put    ("OUTPUT_FLAG      : ");
  put    (NODE.OUTPUT_FLAG);
  new_line(2);
 else
  new_line(2);
  put_line ("No node printed, the value is null. ");
  new_line;
 end if;

end PRINT_NODE;

-------------------------------------------------------------------------------

-- Procedure to print the values of the children nodes

procedure PRINT_CHILDREN_NODES (NODE : in ST_LIST_NODE_PTR) is

 NEXT      : ST_LIST_NODE_PTR;

begin -- PRINT CHILDREN NODES
```

```
if NODE = NULL then
 NEW_LINE;
 PUT_LINE("No value for node passed in, value = null. ");
 NEW_LINE;
else
 NEW_LINE(2);
 PUT_LINE("Inside PRINT CHILDREN NODES procedure.  ");
 PUT_LINE("The parent node is the following:      ");
 NEW_LINE;
 PRINT_NODE(NODE);
 NEW_LINE;
 PUT_LINE("The children nodes are as follows:     ");
 NEW_LINE;
 if NODE.CHILDREN_NODES = NULL then
  PUT_LINE("No children nodes the value is null  ");
 else
  PRINT_NODE(NODE.CHILDREN_NODES);
  NEXT := NODE.CHILDREN_NODES.PREV_ST_PTR;
  while NEXT /= null loop
   PRINT_NODE(NEXT);
   NEXT := NEXT.PREV_ST_PTR;
  end loop;
 end if;
 NEW_LINE;
 PUT_LINE("The above nodes are the children nodes.  ");
end if;

end PRINT_CHILDREN_NODES;

-------------------------------------------------------------------------------

-- Procedure to print the values of the children of children nodes

procedure PRINT_CHILDREN_OF_CHILDREN (NODE : in out ST_LIST_NODE_PTR) is

 NEXT     : ST_LIST_NODE_PTR;
 TEMP     : ST_LIST_NODE_PTR := NODE;

begin  -- PRINT CHILDREN OF CHILDREN

 -- This will advance the pointer to the child of the node passed in
 if NODE /= NULL then
  NODE := NODE.CHILDREN_NODES;
 else
  PUT_LINE("There is no child of the node.  ");
 end if;

 if NODE = NULL then
  NEW_LINE;
  PUT_LINE("No value for node passed in, value = null. ");
  NEW_LINE;
```

```
else
 NEW_LINE(2);
 PUT_LINE("Inside PRINT CHILDREN OF CHILDREN NODES procedure. ");
 PUT_LINE("The parent node is the following:      ");
 NEW_LINE;
 PRINT_NODE(NODE);
 NEW_LINE;
 PUT_LINE("The children nodes are as follows:     ");
 NEW_LINE;
 if NODE.CHILDRFN_NODES = NULL then
  PUT_LINE("No children nodes the value is null  ");
 else
  PRINT_NODE(NODE.CHILDREN_NODES);
  NEXT := NODE.CHILDREN_NODES.PREV_ST_PTR;
  while NEXT /= null loop
   PRINT_NODE(NEXT);
   NEXT := NEXT.PREV_ST_PTR;
  end loop;
 end if;
 NEW_LINE;
 PUT_LINE("The above nodes are the children nodes. ");
 NODE := TEMP;
end if;

end PRINT_CHILDREN_OF_CHILDREN;

-------------------------------------------------------------------------------
-- Procedure to print the values of the children_previous of children nodes

procedure PRINT_CHILDREN_OF_CHILDREN_PREVIOUS (NODE : in out
ST_LIST_NODE_PTR) is

 NEXT     : ST_LIST_NODE_PTR;
 TEMP     : ST_LIST_NODE_PTR := NODE;

begin -- PRINT CHILDREN OF CHILDREN PREVIOUS

 -- This will advance the pointer to the child of the node passed in
 if NODE /= NULL then
  NODE := NODE.CHILDREN_NODES.PREV_ST_PTR;
 else
  PUT_LINE("There is not a second child. ");
 end if;

 if NODE = NULL then
  NEW_LINE;
  PUT_LINE("No value for node passed in, value = null. ");
  NEW_LINE;
 else
  NEW_LINE(2);
  PUT_LINE("Inside PRINT CHILDREN OF CHILDREN_PREVIOUS NODES procedure. ");
```

```
  PUT_LINE("The parent node is the following:      ");
  NEW_LINE;
  PRINT_NODE(NODE);
  NEW_LINE;
  PUT_LINE("The children nodes are as follows:     ");
  NEW_LINE;
  if NODE.CHILDREN_NODES = NULL then
   PUT_LINE("No children nodes the value is null  ");
  else
   PRINT_NODE(NODE.CHILDREN_NODES);
   NEXT := NODE.CHILDREN_NODES.PREV_ST_PTR;
   while NEXT /= null loop
    PRINT_NODE(NEXT);
    NEXT := NEXT.PREV_ST_PTR;
   end loop;
  end if;
  NEW_LINE;
  PUT_LINE("The above nodes are the children nodes.  ");
  NODE := TEMP;
 end if;

end PRINT_CHILDREN_OF_CHILDREN_PREVIOUS;

-------------------------------------------------------------------------------
-- Function to create node for a name

function NAME_TEMPLATE (ID : STRING) return ST_LIST_NODE_PTR  is

 TEMP      : ST_LIST_NODE_PTR;

begin -- NAME_TEMPLATE

 TEMP                 := new ST_LIST_NODE;
 TEMP.ROOT_FAULT(ID'range) := ID;
 return TEMP;

end NAME_TEMPLATE;

-------------------------------------------------------------------------------
-- Function to create node for a value

function VALUE_TEMPLATE (ID : STRING) return ST_LIST_NODE_PTR  is

 TEMP      : ST_LIST_NODE_PTR;

begin -- VALUE_TEMPLATE

 TEMP                 := new ST_LIST_NODE;
 TEMP.ROOT_FAULT(ID'range) := ID;
 return TEMP;
```

```
end VALUE_TEMPLATE;

-----------------------------------------------------------------------------
-- Function to build node for division by zero

function DIVISION_BY_ZERO_TEMPLATE return ST_LIST_NODE_PTR is

 TEMP     : ST_LIST_NODE_PTR;
 TEMP_STR  : STRING (1..23) := "Division By Zero Fault ";

begin -- DIVISION BY ZERO TEMPLATE

 TEMP                  := new ST_LIST_NODE;
 TEMP.ROOT_FAULT(TEMP_STR'range) := TEMP_STR;
 TEMP.TYPE_NODE            := RECTANGLE;
 return TEMP;

end DIVISION_BY_ZERO_TEMPLATE;

-----------------------------------------------------------------------------
-- Function to build node for adding fault

function ADD_FAULT_TEMPLATE return ST_LIST_NODE_PTR is

 TEMP     : ST_LIST_NODE_PTR;
 TEMP_STR  : STRING (1..15) := "Addition Fault ";

begin -- ADDITION FAULT TEMPLATE

 TEMP                  := new ST_LIST_NODE;
 TEMP.ROOT_FAULT(TEMP_STR'range) := TEMP_STR;
 TEMP.TYPE_NODE            := RECTANGLE;
 return TEMP;

end ADD_FAULT_TEMPLATE;

-----------------------------------------------------------------------------
-- Function to build node for subtracting fault

function SUB_FAULT_TEMPLATE return ST_LIST_NODE_PTR is

 TEMP     : ST_LIST_NODE_PTR;
 TEMP_STR  : STRING (1..18) := "Subtraction Fault ";

begin -- SUBTRACTION FAULT TEMPLATE

 TEMP                  := new ST_LIST_NODE;
 TEMP.ROOT_FAULT(TEMP_STR'range) := TEMP_STR;
 TEMP.TYPE_NODE            := RECTANGLE;
 return TEMP;
```

```
end SUB_FAULT_TEMPLATE;

-------------------------------------------------------------------------------
-- Function to build node for adding or subtracting

function ADD_SUB_TEMPLATE (FIRST, SECOND, THIRD : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP     : ST_LIST_NODE_PTR;
 TEMP_STR   : STRING(1..27) := "Addition/Subtraction Fault ";

begin -- ADD SUB TEMPLATE

 TEMP                          := new ST_LIST_NODE;
 TEMP.ROOT_FAULT(TEMP_STR'range)        := TEMP_STR;
 TEMP.NUMBER_OF_CHILDREN             := 3;
 TEMP.TYPE_GATE                  := OR_GATE;
 TEMP.CHILDREN_NODES                 := FIRST;
 TEMP.CHILDREN_NODES.PREV_ST_PTR        := SECOND;
 TEMP.CHILDREN_NODES.PREV_ST_PTR.PREV_ST_PTR := THIRD;
 return TEMP;

end ADD_SUB_TEMPLATE;

-------------------------------------------------------------------------------
-- Function to build node for division

function DIV_MULT_TEMPLATE (FIRST, SECOND, THIRD : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP     : ST_LIST_NODE_PTR;
 TEMP_STR   : STRING(1..32) := "Division / Multiplication Fault ";

begin --DIV MULT TEMPLATE

 TEMP                          := new ST_LIST_NODE;
 TEMP.ROOT_FAULT(TEMP_STR'range)        := TEMP_STR;
 TEMP.NUMBER_OF_CHILDREN             := 3;
 TEMP.TYPE_GATE                  := OR_GATE;
 TEMP.CHILDREN_NODES                 := FIRST;
 TEMP.CHILDREN_NODES.PREV_ST_PTR        := SECOND;
 TEMP.CHILDREN_NODES.PREV_ST_PTR.PREV_ST_PTR := THIRD;

 return TEMP;

end DIV_MULT_TEMPLATE;

-------------------------------------------------------------------------------
-- Function to build node for an index component

function INDEX_COMP_TEMPLATE (FIRST, SECOND : ST_LIST_NODE_PTR)
```

```
 return ST_LIST_NODE_PTR is

 TEMP      : ST_LIST_NODE_PTR;
 TEMP_STR   : STRING(1..24) := "Indexed Component Fault ";

begin -- INDEX COMPONENT TEMPLATE

 TEMP                              := new ST_LIST_NODE;
 TEMP.ROOT_FAULT(TEMP_STR'range)          := TEMP_STR;
 TEMP.NUMBER_OF_CHILDREN                  := 2;
 TEMP.TYPE_GATE                    := OR_GATE;
 TEMP.CHILDREN_NODES                      := FIRST;
 TEMP.CHILDREN_NODES.PREV_ST_PTR          := SECOND;

 return TEMP;

end INDEX_COMP_TEMPLATE;

-------------------------------------------------------------------------------
-- Function to build a node for an assignment statement

function ASSIGNMENT_TEMPLATE (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP         : ST_LIST_NODE_PTR;
 TEMP_CHANGE     : ST_LIST_NODE_PTR;
 TEMP_EXCEPT     : ST_LIST_NODE_PTR;
 TEMP_OP_EVAL    : ST_LIST_NODE_PTR;

 TEMP_STR       : STRING(1..27) := "Assignment Statement Fault ";
 TEMP_CHANGE_STR  : STRING(1..30) := "Change in values caused Fault ";
 TEMP_EXCEPT_STR  : STRING(1..42) := "Exception causes Fault -- Not implemented ";
 TEMP_OP_EVAL_STR : STRING(1..32) := "Operand Evaluation causes Fault ";

begin -- ASSIGNMENT TEMPLATE

 -- Creating Change in values Node
 TEMP_CHANGE                       := new ST_LIST_NODE;
 TEMP_CHANGE.ROOT_FAULT(TEMP_CHANGE_STR'range)  := TEMP_CHANGE_STR;
 TEMP_CHANGE.TYPE_NODE                    := RECTANGLE;

 -- Creating Exception Node
 TEMP_EXCEPT                       := new ST_LIST_NODE;
 TEMP_EXCEPT.ROOT_FAULT(TEMP_EXCEPT_STR'range)  := TEMP_EXCEPT_STR;
 TEMP_CHANGE.TYPE_NODE                    := RECTANGLE;

 -- Creating Operation Evaluation Node
 TEMP_OP_EVAL                      := new ST_LIST_NODE;
 TEMP_OP_EVAL.ROOT_FAULT(TEMP_OP_EVAL_STR'range) :=
TEMP_OP_EVAL_STR;
```

```
if   FIRST /= NULL and SECOND /= NULL then
  TEMP_OP_EVAL.NUMBER_OF_CHILDREN          := 2;
  TEMP_OP_EVAL.TYPE_GATE                   := OR_GATE;
  TEMP_OP_EVAL.CHILDREN_NODES              := FIRST;
  TEMP_OP_EVAL.CHILDREN_NODES.PREV_ST_PTR  := SECOND;
elsif FIRST  = NULL and SECOND /= NULL then
  TEMP_OP_EVAL.NUMBER_OF_CHILDREN          := 1;
  TEMP_OP_EVAL.CHILDREN_NODES              := SECOND;
elsif FIRST /= NULL and SECOND  = NULL then
  TEMP_OP_EVAL.NUMBER_OF_CHILDREN          := 1;
  TEMP_OP_EVAL.CHILDREN_NODES              := FIRST;
end if;

-- Creating Assignment Node
TEMP                                 := new ST_LIST_NODE;
TEMP.ROOT_FAULT(TEMP_STR'range)      := TEMP_STR;
TEMP.NUMBER_OF_CHILDREN              := 3;
TEMP.TYPE_GATE                       := OR_GATE;
TEMP.CHILDREN_NODES                  := TEMP_CHANGE;
TEMP.CHILDREN_NODES.PREV_ST_PTR      := TEMP_EXCEPT;
TEMP.CHILDREN_NODES.PREV_ST_PTR.PREV_ST_PTR  := TEMP_OP_EVAL;

return TEMP;

end ASSIGNMENT_TEMPLATE;
-------------------------------------------------------------------------------
-- Function to build IF node

function IF_STATEMENT_TEMPLATE (FIRST, SECOND, THIRD, FOURTH :
ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP            : ST_LIST_NODE_PTR;
 TEMP_COND_TRUE  : ST_LIST_NODE_PTR;
 TEMP_COND       : ST_LIST_NODE_PTR;
 TEMP_SEQ_STMTS  : ST_LIST_NODE_PTR;
 TEMP_EVAL_COND  : ST_LIST_NODE_PTR;

 TEMP_STR          : STRING(1..26) := "If statement caused fault ";
 TEMP_COND_TRUE_STR  : STRING(1..43) :=
             "Condition true and statements caused fault ";
 TEMP_COND_STR     : STRING(1..18) := "If condition true ";
 TEMP_SEQ_STMTS_STR  : STRING(1..27) := "If statement caused fault ";
 TEMP_EVAL_COND_STR  : STRING(1..37) := "Evaluation of condition caused fault ";

begin -- IF STATEMENT TEMPLATE

 -- Creating the evaluation of the condition node
 TEMP_EVAL_COND                       := new ST_LIST_NODE;
 TEMP_EVAL_COND.ROOT_FAULT(TEMP_EVAL_COND_STR'range) :=
TEMP_EVAL_COND_STR;
```

```
 TEMP_EVAL_COND.CHILDREN_NODES    := FIRST;

 -- Creating node for condition true
 TEMP_COND                        := new ST_LIST_NODE;
 TEMP_COND.ROOT_FAULT(TEMP_COND_STR'range)      := TEMP_COND_STR;

 -- Creating node for statements when condition true
 TEMP_SEQ_STMTS                   := new ST_LIST_NODE;
 TEMP_SEQ_STMTS.ROOT_FAULT(TEMP_SEQ_STMTS_STR'range) :=
TEMP_SEQ_STMTS_STR;
 TEMP_SEQ_STMTS.CHILDREN_NODES                := SECOND;


 -- Creating true condition node for if statement
 TEMP_COND_TRUE                   := new ST_LIST_NODE;
 TEMP_COND_TRUE.ROOT_FAULT(TEMP_COND_TRUE_STR'range) :=
TEMP_COND_TRUE_STR;
 TEMP_COND_TRUE.NUMBER_OF_CHILDREN            := 2;
 TEMP_COND_TRUE.CHILDREN_NODES          := TEMP_COND;
 TEMP_COND_TRUE.CHILDREN_NODES.PREV_ST_PTR    := TEMP_SEQ_STMTS;
 TEMP_COND_TRUE.TYPE_GATE               := AND_GATE;

 -- Creating node for if statement
 TEMP                             := new ST_LIST_NODE;
 TEMP.ROOT_FAULT(TEMP_STR'range)        := TEMP_STR;
 TEMP.TYPE_GATE                   := OR_GATE;
 TEMP.CHILDREN_NODES                  := TEMP_EVAL_COND;
 TEMP.CHILDREN_NODES.PREV_ST_PTR        := TEMP_COND_TRUE;

 if THIRD /= NULL and FOURTH /= NULL then
  TEMP.CHILDREN_NODES.PREV_ST_PTR.PREV_ST_PTR    := THIRD;
  TEMP.CHILDREN_NODES.PREV_ST_PTR.PREV_ST_PTR.PREV_ST_PTR
                          := FOURTH;
  TEMP.NUMBER_OF_CHILDREN             := 4;
 elsif THIRD = NULL and FOURTH /= NULL then
  TEMP.CHILDREN_NODES.PREV_ST_PTR.PREV_ST_PTR    := FOURTH;
  TEMP.NUMBER_OF_CHILDREN             := 3;
 elsif THIRD /= NULL and FOURTH = NULL then
  TEMP.CHILDREN_NODES.PREV_ST_PTR.PREV_ST_PTR    := THIRD;
  TEMP.NUMBER_OF_CHILDREN             := 3;
 else
  TEMP.NUMBER_OF_CHILDREN             := 2;
 end if;

 return TEMP;

end IF_STATEMENT_TEMPLATE;
-------------------------------------------------------------------------------
-- Function to build ELSIF node for IF statement

function ELSIF_TEMPLATE (FIRST, SECOND, THIRD : ST_LIST_NODE_PTR)
```

```
return ST_LIST_NODE_PTR is

 TEMP            : ST_LIST_NODE_PTR;
 TEMP_PREV_COND    : ST_LIST_NODE_PTR;
 TEMP_CUR_FUTURE   : ST_LIST_NODE_PTR;
 TEMP_CUR_ELSIF    : ST_LIST_NODE_PTR;
 TEMP_COND_TRUE    : ST_LIST_NODE_PTR;
 TEMP_SEQ_STMTS    : ST_LIST_NODE_PTR;
 TEMP_OTHER_ELSIF  : ST_LIST_NODE_PTR;
 TEMP_ELSIF_COND   : ST_LIST_NODE_PTR;

 TEMP_STR        : STRING(1..23) := "ELSIF caused the fault ";
 TEMP_PREV_COND_STR : STRING(1..39) := "Previous conditions evaluated to false ";
 TEMP_CUR_FUTURE_STR : STRING(1..38) := "Current/future ELSIF caused the fault ";
 TEMP_CUR_ELSIF_STR : STRING(1..31) := "Current ELSIF caused the fault ";
 TEMP_COND_TRUE_STR : STRING(1..41) := "Current ELSIF condition caused the fault ";
 TEMP_SEQ_STMTS_STR : STRING(1..54) := "Current ELSIF sequence of statements caused
the fault ";
 TEMP_OTHER_ELSIF_STR: STRING(1..31) := "Other ELSIF's caused the fault ";
 TEMP_ELSIF_COND_STR : STRING(1..43) := Evaluatio of Elsif condition caused fault ";

begin -- function ELSIF_TEMPLATE

 -- Creating node for all previous conditions evaluated to false
 TEMP_PREV_COND                          := new ST_LIST_NODE;
 TEMP_PREV_COND.ROOT_FAULT(TEMP_PREV_COND_STR'range)
:=TEMP_PREV_COND_STR;
 TEMP_PREV_COND.TYPE_NODE                := RECTANGLE;

 -- Creating node for current ELSIF condition evaluating to true
 TEMP_COND_TRUE                          := new ST_LIST_NODE;
 TEMP_COND_TRUE.ROOT_FAULT(TEMP_COND_TRUE_STR'range) :=
TEMP_COND_TRUE_STR;

 TEMP_ELSIF_COND                                 := new ST_LIST_NODE_PTR;
 TEMP_ELSIF_COND.ROOT_FAULT(TEMP_ELSIF_COND_STR'range) :=
TEMP_ELSIF_COND_STR;
 TEMP_ELISF_COND.CHILDREN_NODES          := FIRST;

 -- Creating node for current ELSIF sequence of statements
 TEMP_SEQ_STMTS                          := new ST_LIST_NODE;
 TEMP_SEQ_STMTS.ROOT_FAULT(TEMP_SEQ_STMTS_STR'range) :=
TEMP_SEQ_STMTS_STR;
 TEMP_SEQ_STMTS.CHILDREN_NODES           := SECOND;

 -- Creating node for current ELSIF causing fault
 TEMP_CUR_ELSIF                          := new ST_LIST_NODE;
 TEMP_CUR_ELSIF.ROOT_FAULT(TEMP_CUR_ELSIF_STR'range) :=
TEMP_CUR_ELSIF_STR;
 TEMP_CUR_ELSIF.NUMBER_OF_CHILDREN       := 2;
 TEMP_CUR_ELSIF.CHILDREN_NODES           := TEMP_COND_TRUE;
```

```
 TEMP_CUR_ELSIF.CHILDREN_NODES.PREV_ST_PTR      := TEMP_SEQ_STMTS;
 TEMP_CUR_ELSIF.TYPE_GATE                       := AND_GATE;

 -- Creating node for other ELSIF's possibly causing fault
 TEMP_OTHER_ELSIF                               := new ST_LIST_NODE;
 TEMP_OTHER_ELSIF.ROOT_FAULT(TEMP_OTHER_ELSIF_STR'range) :=
TEMP_OTHER_ELSIF_STR;
 TEMP_OTHER_ELSIF.CHILDREN_NODES                := THIRD;

 -- Creating node for current or previous ELSIF causing the fault
 TEMP_CUR_FUTURE                                := new ST_LIST_NODE;
 TEMP_CUR_FUTURE.ROOT_FAULT(TEMP_CUR_FUTURE_STR'range) :=
TEMP_CUR_FUTURE_STR;
 TEMP_CUR_FUTURE.TYPE_GATE                      := OR_GATE;
 TEMP_CUR_FUTURE.NUMBER_OF_CHILDREN             := 3;
 TEMP_CUR_FUTURE.CHILDREN_NODES                 := TEMP_ELSIF_COND;
 TEMP_CUR_FUTURE.CHILDREN_NODES.PREV_ST_PTR     := TEMP_CUR_ELISIF;
 TEMP_CUR_FUTURE.CHILDREN_NODES.PREV_ST_PTR.PREV_ST_PTR :=
TEMP_OTHER_ELSIF;

 -- Creating node for the ELSIF statements
 TEMP                           := new ST_LIST_NODE;
 TEMP.ROOT_FAULT(TEMP_STR'range)    := TEMP_STR;
 TEMP.NUMBER_OF_CHILDREN        := 2;
 TEMP.TYPE_GATE                 := AND_GATE;
 TEMP.CHILDREN_NODES            := TEMP_PREV_COND;
 TEMP.CHILDREN_NODES.PREV_ST_PTR    := TEMP_CUR_FUTURE;

 return TEMP;

end ELSIF_TEMPLATE;
-------------------------------------------------------------------------------
-- Function to build ELSE node for IF statement

function ELSE_TEMPLATE (FIRST : ST_LIST_NODE_PTR) return ST_LIST_NODE_PTR is


 TEMP        : ST_LIST_NODE_PTR;
 TEMP_COND   : ST_LIST_NODE_PTR;
 TEMP_STMTS  : ST_LIST_NODE_PTR;

 TEMP_STR      : STRING(1..23) := "ELSE part caused Fault ";
 TEMP_COND_STR : STRING(1..61) :=
          "IF condition and ELSIF condition, if any, evaluated to false ";
 TEMP_STMTS_STR : STRING(1..32) := "Statements in ELSE caused Fault ";

begin -- ELSE TEMPLATE

 -- Creating condition node for else statement
 TEMP_COND                      := new ST_LIST_NODE;
 TEMP_COND.ROOT_FAULT(TEMP_COND_STR'range) := TEMP_COND_STR;
```

```
 TEMP_COND.TYPE_NODE                    := 3;

 -- Creating statements node for else statement
 TEMP_STMTS                             := new ST_LIST_NODE;
 TEMP_STMTS.ROOT_FAULT(TEMP_STMTS_STR'range) := TEMP_STMTS_STR;
 TEMP_STMTS.CHILDREN_NODES              := FIRST;

 -- Creating Else node for else
 TEMP                                   := new ST_LIST_NODE;
 TEMP.ROOT_FAULT(TEMP_STR'range)        := TEMP_STR;
 TEMP.NUMBER_OF_CHILDREN                := 2;
 TEMP.TYPE_GATE                         := 1;
 TEMP.CHILDREN_NODES                    := TEMP_COND;
 TEMP.CHILDREN_NODES.PREV_ST_PTR        := TEMP_STMTS;

 return TEMP;

end ELSE_TEMPLATE;
-------------------------------------------------------------------------------
-- Function to build nodes for sequence of statements ..STMT..

function STATEMENTS (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP           : ST_LIST_NODE_PTR;
 TEMP_LAST      : ST_LIST_NODE_PTR;
 TEMP_PREV      : ST_LIST_NODE_PTR;
 TEMP_LAST_MASK : ST_LIST_NODE_PTR;
 TEMP_SEQ_PRIOR : ST_LIST_NODE_PTR;

 TEMP_STR          : STRING(1..36) := "Sequence of statements caused fault ";
 TEMP_LAST_STR     : STRING(1..36) := "Last statement caused fault       ";
 TEMP_PREV_STR     : STRING(1..36) := "Previous statements caused fault  ";
 TEMP_LAST_MASK_STR : STRING(1..36) := "Last Statement did not mask fault ";
 TEMP_SEQ_PRIOR_STR : STRING(1..36) := "Sequence prior to last caused fault ";

 NEXT           : ST_LIST_NODE_PTR;
 ADD_LOC        : ST_LIST_NODE_PTR;
 END_NODE       : BOOLEAN := FALSE;

begin -- function Statements

 -- Creating nodes for template of sequence of statements
 -- Building node for last statement did not mask fault
 TEMP_LAST_MASK                         := new ST_LIST_NODE;
 TEMP_LAST_MASK.ROOT_FAULT(TEMP_LAST_MASK_STR'range) :=
TEMP_LAST_MASK_STR;

 -- Building node for sequence prior to last statement caused fault
 TEMP_SEQ_PRIOR                         := new ST_LIST_NODE;
```

```
 TEMP_SEQ_PRIOR.ROOT_FAULT(TEMP_SEQ_PRIOR_STR'range) :=
TEMP_SEQ_PRIOR_STR;

-- Building node for previous statements caused the fault
TEMP_PREV                              := new ST_LIST_NODE;
TEMP_PREV.ROOT_FAULT(TEMP_PREV_STR'range)      := TEMP_PREV_STR;
TEMP_PREV.TYPE_GATE                    := AND_GATE;
TEMP_PREV.NUMBER_OF_CHILDREN           := 2;
TEMP_PREV.CHILDREN_NODES               := TEMP_LAST_MASK;
TEMP_PREV.CHILDREN_NODES.PREV_ST_PTR   := TEMP_SEQ_PRIOR;

-- Building node for the last statement caused the fault
TEMP_LAST                              := new ST_LIST_NODE;
TEMP_LAST.ROOT_FAULT(TEMP_LAST_STR'range)      := TEMP_LAST_STR;

-- Building node for the sequence caused the fault
TEMP                                   := new ST_LIST_NODE;
TEMP.ROOT_FAULT(TEMP_STR'range)        := TEMP_STR;
TEMP.TYPE_GATE                         := OR_GATE;
TEMP.NUMBER_OF_CHILDREN                := 2;
TEMP.CHILDREN_NODES                    := TEMP_LAST;
TEMP.CHILDREN_NODES.PREV_ST_PTR        := TEMP_PREV;

-- If both ..stmt.. and stmt are null
if FIRST = NULL and SECOND = NULL then
 return null;
-- If ..stmt.. is null and stmt is not null
elsif FIRST = NULL and SECOND /= NULL then
 TEMP_LAST.CHILDREN_NODES    := SECOND;
 TEMP_LAST.NUMBER_OF_CHILDREN := 1;
 return TEMP;
-- If ..stmt.. is not null but stmt is null
elsif FIRST /= NULL and SECOND = NULL then
 TEMP_LAST.CHILDREN_NODES    := FIRST;
 TEMP_LAST.NUMBER_OF_CHILDREN := 1;
 return TEMP;
-- Both ..stmt.. and stmt have a valid statement
else
 -- Initialize temporary pointers to help place the statements
 NEXT   :=
FIRST.CHILDREN_NODES.PREV_ST_PTR.CHILDREN_NODES.PREV_ST_PTR.CHILDR
EN_NODES;
 ADD_LOC :=
FIRST.CHILDREN_NODES.PREV_ST_PTR.CHILDREN_NODES.PREV_ST_PTR;

 -- Loop until you find the place to insert the nodes
 while NOT END_NODE loop
  if NEXT = NULL then
   ADD_LOC.CHILDREN_NODES          := TEMP;
   ADD_LOC.NUMBER_OF_CHILDREN      := 1;
   TEMP.CHILDREN_NODES.CHILDREN_NODES   := SECOND;
```

```
    TEMP.CHILDREN_NODES.NUMBER_OF_CHILDREN := 1;
    END_NODE                    := TRUE;
    return FIRST;
   else
    ADD_LOC :=
NEXT.CHILDREN_NODES.PREV_ST_PTR.CHILDREN_NODES.PREV_ST_PTR;
    NEXT   :=
NEXT.CHILDREN_NODES.PREV_ST_PTR.CHILDREN_NODES.PREV_ST_PTR.CHILDR
EN_NODES;
   end if;
  end loop;
 end if;

end STATEMENTS;
-------------------------------------------------------------------
-- Function to build node joining elements of a relation

function BUILD_REL (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP       : ST_LIST_NODE_PTR;
 TEMP_STR   : string(1..26) := "Relation caused the fault ";

begin -- Function BUILD REL

 TEMP                  := new ST_LIST_NODE;
 TEMP.ROOT_FAULT(TEMP_STR'range) := TEMP_STR;
 if SECOND /= NULL then
  TEMP.TYPE_GATE           := OR_GATE;
  TEMP.NUMBER_OF_CHILDREN     := 2;
  TEMP.CHILDREN_NODES         := FIRST;
  TEMP.CHILDREN_NODES.PREV_ST_PTR := SECOND;
 else
  TEMP.NUMBER_OF_CHILDREN     := 1;
  TEMP.CHILDREN_NODES         := FIRST;
 end if;

 return TEMP;

end BUILD_REL;

-------------------------------------------------------------------
-- Function to build and relation node

function BUILD_REL_AND (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP       : ST_LIST_NODE_PTR;
 TEMP_LEFT     : ST_LIST_NODE_PTR;
 TEMP_RIGHT    : ST_LIST_NODE_PTR;
```

```
 TEMP_STR       : STRING(1..30) := "And Relation caused the fault ";
 TEMP_LEFT_STR  : STRING(1..36) := "Left or Prior relation caused fault ";
 TEMP_RIGHT_STR : STRING(1..28) := "Right relation caused fault ";

begin -- Beginning of the REL AND template

 -- Building left node for and relation
 TEMP_LEFT                          := new ST_LIST_NODE;
 TEMP_LEFT.ROOT_FAULT(TEMP_LEFT_STR'range)   := TEMP_LEFT_STR;
 TEMP_LEFT.NUMBER_OF_CHILDREN               := 1;
 TEMP_LEFT.CHILDREN_NODES                := FIRST;

 -- Building right node for and relation
 TEMP_RIGHT                         := new ST_LIST_NODE;
 TEMP_RIGHT.ROOT_FAULT(TEMP_RIGHT_STR'range) := TEMP_RIGHT_STR;
 TEMP_RIGHT.NUMBER_OF_CHILDREN              := 1;
 TEMP_RIGHT.CHILDREN_NODES               := SECOND;

 -- Building and relation node
 TEMP                               := new ST_LIST_NODE;
 TEMP.ROOT_FAULT(TEMP_STR'range)            := TEMP_STR;
 TEMP.NUMBER_OF_CHILDREN                    := 2;
 TEMP.TYPE_GATE                          := OR_GATE;
 TEMP.CHILDREN_NODES                     := TEMP_LEFT;
 TEMP.CHILDREN_NODES.PREV_ST_PTR            := TEMP_RIGHT;

 RETURN TEMP;

end BUILD_REL_AND;

-----------------------------------------------------------------------------------------------------------
-- Function to build or relation node

function BUILD_REL_OR (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP       : ST_LIST_NODE_PTR;
 TEMP_LEFT  : ST_LIST_NODE_PTR;
 TEMP_RIGHT : ST_LIST_NODE_PTR;

 TEMP_STR       : STRING(1..29) := "Or Relation caused the fault ";
 TEMP_LEFT_STR  : STRING(1..36) := "Left or Prior relation caused fault ";
 TEMP_RIGHT_STR : STRING(1..28) := "Right relation caused fault ";

begin -- Beginning of the REL AND template

 -- Building left node for and relation
 TEMP_LEFT                          := new ST_LIST_NODE;
 TEMP_LEFT.ROOT_FAULT(TEMP_LEFT_STR'range)   := TEMP_LEFT_STR;
 TEMP_LEFT.NUMBER_OF_CHILDREN               := 1;
 TEMP_LEFT.CHILDREN_NODES                := FIRST;
```

```
-- Building right node for and relation
TEMP_RIGHT                          := new ST_LIST_NODE;
TEMP_RIGHT.ROOT_FAULT(TEMP_RIGHT_STR'range)  := TEMP_RIGHT_STR;
TEMP_RIGHT.NUMBER_OF_CHILDREN               := 1;
TEMP_RIGHT.CHILDREN_NODES                   := SECOND;

-- Building and relation node
TEMP                                := new ST_LIST_NODE;
TEMP.ROOT_FAULT(TEMP_STR'range)             := TEMP_STR;
TEMP.NUMBER_OF_CHILDREN                     := 2;
TEMP.TYPE_GATE                      := OR_GATE;
TEMP.CHILDREN_NODES                         := TEMP_LEFT;
TEMP.CHILDREN_NODES.PREV_ST_PTR             := TEMP_RIGHT;

RETURN TEMP;

end BUILD_REL_OR;

-------------------------------------------------------------------------------

-- Function to build and then relation node

function BUILD_REL_AND_THEN (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP          : ST_LIST_NODE_PTR;
 TEMP_LEFT     : ST_LIST_NODE_PTR;
 TEMP_RIGHT    : ST_LIST_NODE_PTR;
 TEMP_R_L      : ST_LIST_NODE_PTR;
 TEMP_R_R      : ST_LIST_NODE_PTR;

 TEMP_STR      : STRING(1..35) := "And Then Relation caused the fault ";
 TEMP_LEFT_STR : STRING(1..36) := "Left or Prior relation caused fault ";
 TEMP_RIGHT_STR: STRING(1..28) := "Right relation caused fault ";
 TEMP_R_L_STR  : STRING(1..19) := "Left relation true ";
 TEMP_R_R_STR  : STRING(1..42) := "Evaluation of right relation caused fault ";

begin -- Beginning of the REL AND THEN template

 -- Building right node's left child
 TEMP_R_L                           := new ST_LIST_NODE;
 TEMP_R_L.ROOT_FAULT(TEMP_R_L_STR'range)     := TEMP_R_L_STR;

 -- Building right node's right child
 TEMP_R_R                           := new ST_LIST_NODE;
 TEMP_R_R.ROOT_FAULT(TEMP_R_R_STR'range)     := TEMP_R_R_STR;
 TEMP_R_R.NUMBER_OF_CHILDREN                := 1;
 TEMP_R_R.CHILDREN_NODES                    := SECOND;

 -- Building right node for and relation
 TEMP_RIGHT                         := new ST_LIST_NODE;
```

```
TEMP_RIGHT.ROOT_FAULT(TEMP_RIGHT_STR'range)  := TEMP_RIGHT_STR;
TEMP_RIGHT.NUMBER_OF_CHILDREN            := 2;
TEMP_RIGHT.TYPE_GATE                     := AND_GATE;
TEMP_RIGHT.CHILDREN_NODES                := TEMP_R_L;
TEMP_RIGHT.CHILDREN_NODES.PREV_ST_PTR    := TEMP_R_R;

-- Building left node for and relation
TEMP_LEFT                        := new ST_LIST_NODE;
TEMP_LEFT.ROOT_FAULT(TEMP_LEFT_STR'range)   := TEMP_LEFT_STR;
TEMP_LEFT.NUMBER_OF_CHILDREN             := 1;
TEMP_LEFT.CHILDREN_NODES                 := FIRST;

-- Building and relation node
TEMP                             := new ST_LIST_NODE;
TEMP.ROOT_FAULT(TEMP_STR'range)          := TEMP_STR;
TEMP.NUMBER_OF_CHILDREN                  := 2;
TEMP.TYPE_GATE                   := OR_GATE;
TEMP.CHILDREN_NODES                      := TEMP_LEFT;
TEMP.CHILDREN_NODES.PREV_ST_PTR          := TEMP_RIGHT;

RETURN TEMP;

end BUILD_REL_AND_THEN;

-------------------------------------------------------------------------------
-- Function to build or else relation node

function BUILD_REL_OR_ELSE (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP          : ST_LIST_NODE_PTR;
 TEMP_LEFT     : ST_LIST_NODE_PTR;
 TEMP_RIGHT    : ST_LIST_NODE_PTR;
 TEMP_R_L      : ST_LIST_NODE_PTR;
 TEMP_R_R      : ST_LIST_NODE_PTR;

 TEMP_STR      : STRING(1..34) := "Or else Relation caused the fault ";
 TEMP_LEFT_STR : STRING(1..36) := "Left or Prior relation caused fault ";
 TEMP_RIGHT_STR: STRING(1..28) := "Right relation caused fault ";
 TEMP_R_L_STR  : STRING(1..20) := "Left relation false ";
 TEMP_R_R_STR  : STRING(1..42) := "Evaluation of right relation caused fault ";

begin -- Beginning of the REL AND THEN template

 -- Building right node's left child
 TEMP_R_L                        := new ST_LIST_NODE;
 TEMP_R_L.ROOT_FAULT(TEMP_R_L_STR'range)  := TEMP_R_L_STR;

 -- Building right node's right child
 TEMP_R_R                        := new ST_LIST_NODE;
 TEMP_R_R.ROOT_FAULT(TEMP_R_R_STR'range)  := TEMP_R_R_STR;
```

```
TEMP_R_R.NUMBER_OF_CHILDREN          := 1;
TEMP_R_R.CHILDREN_NODES              := SECOND;

-- Building right node for and relation
TEMP_RIGHT                           := new ST_LIST_NODE;
TEMP_RIGHT.ROOT_FAULT(TEMP_RIGHT_STR'range)  := TEMP_RIGHT_STR;
TEMP_RIGHT.NUMBER_OF_CHILDREN        := 2;
TEMP_RIGHT.TYPE_GATE                 := AND_GATE;
TEMP_RIGHT.CHILDREN_NODES            := TEMP_R_L;
TEMP_RIGHT.CHILDREN_NODES.PREV_ST_PTR    := TEMP_R_R;

-- Building left node for and relation
TEMP_LEFT                            := new ST_LIST_NODE;
TEMP_LEFT.ROOT_FAULT(TEMP_LEFT_STR'range)   := TEMP_LEFT_STR;
TEMP_LEFT.NUMBER_OF_CHILDREN         := 1;
TEMP_LEFT.CHILDREN_NODES             := FIRST;

-- Building and relation node
TEMP                                 := new ST_LIST_NODE;
TEMP.ROOT_FAULT(TEMP_STR'range)      := TEMP_STR;
TEMP.NUMBER_OF_CHILDREN              := 2;
TEMP.TYPE_GATE                       := OR_GATE;
TEMP.CHILDREN_NODES                  := TEMP_LEFT;
TEMP.CHILDREN_NODES.PREV_ST_PTR      := TEMP_RIGHT;

RETURN TEMP;

end BUILD_REL_OR_ELSE;

-------------------------------------------------------------------------------
-- Function to build node for range name specified causing fault

function BUILD_RANGE_NAME (FIRST : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP       : ST_LIST_NODE_PTR;
 TEMP_STR   : STRING(1..29) := "Range specified caused fault ";

begin -- Beginning of function to build range specified node

 TEMP                        := new ST_LIST_NODE;
 TEMP.ROOT_FAULT(TEMP_STR'range) := TEMP_STR;
 TEMP.CHILDREN_NODES         := FIRST;
 TEMP.NUMBER_OF_CHILDREN     := 1;

 RETURN TEMP;

end BUILD_RANGE_NAME;

-------------------------------------------------------------------------------
-- Function to build node for range specified causing fault
```

```
function BUILD_RNG (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP       : ST_LIST_NODE_PTR;
 TEMP_STR   : STRING(1..29) := "Range specified caused fault ";

begin -- Beginning of function to build range specified node

 TEMP                    := new ST_LIST_NODE;
 TEMP.ROOT_FAULT(TEMP_STR'range) := TEMP_STR;
 TEMP.CHILDREN_NODES     := FIRST;
 TEMP.CHILDREN_NODES.PREV_ST_PTR := SECOND;
 TEMP.NUMBER_OF_CHILDREN := 2;
 TEMP.TYPE_GATE          := OR_GATE;

 RETURN TEMP;

end BUILD_RNG;

-------------------------------------------------------------------------------

-- Function to build node for discreet range 1 child causing fault

function BUILD_DESCR_RNG_1 (FIRST : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP       : ST_LIST_NODE_PTR;
 TEMP_STR   : STRING(1..28) := "Discreet range caused fault ";

begin -- Function to build discreet range node causing fault

 TEMP                    := new ST_LIST_NODE;
 TEMP.ROOT_FAULT(TEMP_STR'range) := TEMP_STR;
 TEMP.CHILDREN_NODES     := FIRST;
 TEMP.NUMBER_OF_CHILDREN := 1;

 RETURN TEMP;

end BUILD_DESCR_RNG_1;

-------------------------------------------------------------------------------

-- Function to build node for discreet range 2 children causing fault

function BUILD_DESCR_RNG_2 (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP       : ST_LIST_NODE_PTR;
 TEMP_STR   : STRING(1..28) := "Discreet range caused fault ";

begin -- Function to build discreet range node causing fault
```

```
 TEMP                    := new ST_LIST_NODE;
 TEMP.ROOT_FAULT(TEMP_STR'range) := TEMP_STR;
 TEMP.CHILDREN_NODES     := FIRST;
 TEMP.CHILDREN_NODES.PREV_ST_PTR := SECOND;
 TEMP.NUMBER_OF_CHILDREN := 2;
 TEMP.TYPE_GATE          := OR_GATE;

 RETURN TEMP;

end BUILD_DESCR_RNG_2;

-------------------------------------------------------------------------------
-- Function to build node for reverse specification causing fault

function BUILD_REVERSE_NODE
 return ST_LIST_NODE_PTR is

 TEMP     : ST_LIST_NODE_PTR;
 TEMP_STR : STRING(1..35) := "Reverse specification caused fault ";

begin -- Function to build discreet range node causing fault

 TEMP                    := new ST_LIST_NODE;
 TEMP.ROOT_FAULT(TEMP_STR'range) := TEMP_STR;

 RETURN TEMP;

end BUILD_REVERSE_NODE;

-------------------------------------------------------------------------------
-- Function to build nodes for the loop parameter causing fault

function BUILD_LOOP_PRM_SPEC (FIRST, SECOND, THIRD : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP     : ST_LIST_NODE_PTR;
 TEMP_ID  : ST_LIST_NODE_PTR;

 TEMP_STR    : STRING(1..37) := "Parameter specification caused fault ";
 TEMP_ID_STR : STRING(1..34) := "Identifier specified caused fault ";

begin -- Function to build loop parameter template

 -- Building left child where name of range specified
 TEMP_ID                 := new ST_LIST_NODE;
 TEMP_ID.ROOT_FAULT(TEMP_ID_STR'range) := TEMP_ID_STR;
 TEMP_ID.NUMBER_OF_CHILDREN := 1;
 TEMP_ID.CHILDREN_NODES     := FIRST;

 -- Building loop parameter specification node
 TEMP                    := new ST_LIST_NODE;
```

```
 TEMP.ROOT_FAULT(TEMP_STR'range) := TEMP_STR;
 TEMP.CHILDREN_NODES          := TEMP_ID;
 if SECOND /= NULL then
  TEMP.CHILDREN_NODES.PREV_ST_PTR          := SECOND;
  TEMP.CHILDREN_NODES.PREV_ST_PTR.PREV_ST_PTR := THIRD;
  TEMP.NUMBER_OF_CHILDREN                  := 3;
 else
  TEMP.CHILDREN_NODES.PREV_ST_PTR          := THIRD;
  TEMP.NUMBER_OF_CHILDREN                  := 2;
 end if;
 TEMP.TYPE_GATE               := OR_GATE;

 RETURN TEMP;

end BUILD_LOOP_PRM_SPEC;

-------------------------------------------------------------------------------
-- Function to build for loop iteration node

function BUILD_FOR_ITERATION (FIRST : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP    : ST_LIST_NODE_PTR;
 TEMP_STR : STRING(1..30) := "Iteration scheme caused fault ";

begin -- Function to build for loop node

 TEMP                    := new ST_LIST_NODE;
 TEMP.ROOT_FAULT(TEMP_STR'range) := TEMP_STR;
 TEMP.NUMBER_OF_CHILDREN      := 1;
 TEMP.CHILDREN_NODES          := FIRST;
 TEMP.TYPE_GATE               := OR_GATE;

 RETURN TEMP;

end BUILD_FOR_ITERATION;

-------------------------------------------------------------------------------
-- Function to build while loop iteration node

function BUILD_WHILE_ITERATION (FIRST : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP    : ST_LIST_NODE_PTR;
 TEMP_COND : ST_LIST_NODE_PTR;

 TEMP_STR    : STRING(1..30) := "Iteration scheme caused fault ";
 TEMP_COND_STR : STRING(1..41) := "Evaluation of condition caused the fault ";

begin -- Function to build while loop node
```

```
-- Building while condition node
TEMP_COND                       := new ST_LIST_NODE;
TEMP_COND.ROOT_FAULT(TEMP_COND_STR'range) := TEMP_COND_STR;
TEMP_COND.NUMBER_OF_CHILDREN        := 1;
TEMP_COND.CHILDREN_NODES            := FIRST;

-- Building iteration node
TEMP                    := new ST_LIST_NODE;
TEMP.ROOT_FAULT(TEMP_STR'range) := TEMP_STR;
TEMP.NUMBER_OF_CHILDREN     := 1;
TEMP.CHILDREN_NODES         := TEMP_COND;
TEMP.TYPE_GATE              := OR_GATE;

RETURN TEMP;

end BUILD_WHILE_ITERATION;
-------------------------------------------------------------------------------
-- Function to build loop statement template

function BUILD_LOOP_STMT (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP       : ST_LIST_NODE_PTR;
 TEMP_2     : ST_LIST_NODE_PTR;
 TEMP_3     : ST_LIST_NODE_PTR;
 TEMP_4     : ST_LIST_NODE_PTR;
 TEMP_5     : ST_LIST_NODE_PTR;
 TEMP_7     : ST_LIST_NODE_PTR;
 TEMP_8     : ST_LIST_NODE_PTR;
 TEMP_11    : ST_LIST_NODE_PTR;
 TEMP_12    : ST_LIST_NODE_PTR;

 TEMP_STR   : STRING(1..28) := "Loop Statement caused fault ";
 TEMP_2_STR : STRING(1..20) := "Loop never executed ";
 TEMP_3_STR : STRING(1..39) := "Loop condition evaluation caused fault ";
 TEMP_4_STR : STRING(1..27) := "Nth Iteration caused fault ";
 TEMP_5_STR : STRING(1..24) := "Wrong Type of loop used ";
 TEMP_7_STR : STRING(1..36) := "Sequence of statements caused fault ";
 TEMP_8_STR : STRING(1..24) := "Condition true past n-1 ";
 TEMP_11_STR : STRING(1..32) := "Condition true at n-1 iteration ";
 TEMP_12_STR : STRING(1..43) := "Sequence of statements kept condition true ";

begin -- Beginning of function to build loop statement template

 -- Building node 12
 TEMP_12                    := new ST_LIST_NODE;
 TEMP_12.ROOT_FAULT(TEMP_12_STR'range) := TEMP_12_STR;
 TEMP_12.NUMBER_OF_CHILDREN     := 1;
 TEMP_12.CHILDREN_NODES         := SECOND;

 -- Building node 11
```

```
TEMP_11 := new ST_LIST_NODE;
TEMP_11.ROOT_FAULT(TEMP_11_STR'range) := TEMP_11_STR;

-- Building node 8
TEMP_8                    := new ST_LIST_NODE;
TEMP_8.ROOT_FAULT(TEMP_8_STR'range) := TEMP_8_STR;
TEMP_8.NUMBER_OF_CHILDREN      := 2;
TEMP_8.CHILDREN_NODES          := TEMP_11;
TEMP_8.CHILDREN_NODES.PREV_ST_PTR  := TEMP_12;
TEMP_8.TYPE_GATE               := AND_GATE;

-- Building node 7
TEMP_7                    := new ST_LIST_NODE;
TEMP_7.ROOT_FAULT(TEMP_7_STR'range) := TEMP_7_STR;
TEMP_7.NUMBER_OF_CHILDREN      := 1;
TEMP_7.CHILDREN_NODES          := SECOND;

-- Building node 4
TEMP_4                    := new ST_LIST_NODE;
TEMP_4.ROOT_FAULT(TEMP_4_STR'range) := TEMP_4_STR;
TEMP_4.NUMBER_OF_CHILDREN      := 2;
TEMP_4.CHILDREN_NODES          := TEMP_7;
TEMP_4.CHILDREN_NODES.PREV_ST_PTR  := TEMP_8;
TEMP_4.TYPE_GATE               := AND_GATE;

-- Building node 5
TEMP_5                    := new ST_LIST_NODE;
TEMP_5.ROOT_FAULT(TEMP_5_STR'range) := TEMP_5_STR;

-- Building node 3
TEMP_3                    := new ST_LIST_NODE;
TEMP_3.ROOT_FAULT(TEMP_3_STR'range) := TEMP_3_STR;
TEMP_3.TYPE_GATE               := OR_GATE;
TEMP_3.CHILDREN_NODES          := TEMP_5;
if FIRST /= NULL then
 TEMP_3.CHILDREN_NODES.PREV_ST_PTR := FIRST;
 TEMP_3.NUMBER_OF_CHILDREN     := 2;
else
 TEMP_3.NUMBER_OF_CHILDREN     := 1;
end if;

-- Building node 2
TEMP_2                    := new ST_LIST_NODE;
TEMP_2.ROOT_FAULT(TEMP_2_STR'range) := TEMP_2_STR;

-- Building node 1 loop statement template
TEMP                           := new ST_LIST_NODE;
TEMP.ROOT_FAULT(TEMP_STR'range)     := TEMP_STR;
TEMP.TYPE_GATE                 := OR_GATE;
TEMP.CHILDREN_NODES            := TEMP_2;
TEMP.CHILDREN_NODES.PREV_ST_PTR     := TEMP_3;
```

```
 TEMP.CHILDREN_NODES.PREV_ST_PTR.PREV_ST_PTR := TEMP_4;
 TEMP.NUMBER_OF_CHILDREN              := 3;


 RETURN TEMP;

end BUILD_LOOP_STMT;

-------------------------------------------------------------------------------
-- Function to build case statement template

function BUILD_CASE_STMT_TEMPLATE (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP     : ST_LIST_NODE_PTR;
 TEMP_2   : ST_LIST_NODE_PTR;

 TEMP_STR  : STRING(1..28) := "Case statement caused fault ";
 TEMP_2_STR : STRING(1..38) := "Evaluation of expression caused fault ";

begin -- BUILD_CASE_STMT_TEMPLATE

 -- Building evaluation of expression node
 TEMP_2                    := new ST_LIST_NODE;
 TEMP_2.ROOT_FAULT(TEMP_2_STR'range) := TEMP_2_STR;
 TEMP_2.NUMBER_OF_CHILDREN      := 1;
 TEMP_2.CHILDREN_NODES          := FIRST;

 -- Building Case stmt node
 TEMP                      := new ST_LIST_NODE;
 TEMP.ROOT_FAULT(TEMP_STR'range)  := TEMP_STR;
 TEMP.TYPE_GATE             := OR_GATE;
 TEMP.NUMBER_OF_CHILDREN        := 2;
 TEMP.CHILDREN_NODES            := TEMP_2;
 TEMP.CHILDREN_NODES.PREV_ST_PTR  := SECOND;

 RETURN TEMP;

end BUILD_CASE_STMT_TEMPLATE;

-------------------------------------------------------------------------------
-- Function to build at least one or more case alternatives templates

function BUILD_CASE_ALT_1_MORE (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP            : ST_LIST_NODE_PTR;
 TEMP_STR          : STRING(1..30) := "Case alternative caused fault ";

 NEXT            : ST_LIST_NODE_PTR;
 ADD_LOC           : ST_LIST_NODE_PTR;
```

```
 END_NODE            : BOOLEAN := FALSE;

begin  -- Build case alt 1 more

 TEMP                      := new ST_LIST_NODE;
 TEMP.ROOT_FAULT(TEMP_STR'range) := TEMP_STR;
 TEMP.NUMBER_OF_CHILDREN       := 1;

 if SECOND = NULL then        -- Only one case alternative
  TEMP.CHILDREN_NODES := FIRST;
 else                    -- Multiple case alternatives ( > 2 )
  TEMP.CHILDREN_NODES := SECOND;

  -- Initialize temporary variables
  NEXT   := SECOND.CHILDREN_NODES.PREV_ST_PTR.CHILDREN_NODES;
  ADD_LOC := SECOND.CHILDREN_NODES.PREV_ST_PTR;

  -- Iteration to find location to add child
  while NOT END_NODE loop
   if NEXT = NULL then
    ADD_LOC.CHILDREN_NODES := FIRST;
    END_NODE          := TRUE;
   else
    ADD_LOC := NEXT.CHILDREN_NODES.PREV_ST_PTR;
    NEXT   := NEXT.CHILDREN_NODES.PREV_ST_PTR.CHILDREN_NODES;
   end if;      -- next = null
  end loop;      -- not end_node
 end if;         -- second = null

 RETURN TEMP;

end BUILD_CASE_ALT_1_MORE;
-------------------------------------------------------------------------------
-- Function to build 0 or more case alternatives template

function BUILD_CASE_ALT_0_MORE (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP      : ST_LIST_NODE_PTR;
 TEMP_9    : ST_LIST_NODE_PTR;

 TEMP_STR    : STRING(1..42) := "Current/previous alternative caused fault ";
 TEMP_9_STR  : STRING(1..34) := "Previous alternative caused fault ";

begin -- BUILD CASE ALT 0 MORE

 TEMP_9                    := new ST_LIST_NODE;
 TEMP_9.ROOT_FAULT(TEMP_9_STR'range) := TEMP_9_STR;
 TEMP_9.NUMBER_OF_CHILDREN       := 1;
 TEMP_9.CHILDREN_NODES         := FIRST;
```

```
TEMP                        := new ST_LIST_NODE;
TEMP.ROOT_FAULT(TEMP_STR'range)   := TEMP_STR;
TEMP.TYPE_GATE              := OR_GATE;
TEMP.NUMBER_OF_CHILDREN     := 2;
TEMP.CHILDREN_NODES         := SECOND;
TEMP.CHILDREN_NODES.PREV_ST_PTR   := TEMP_9;

RETURN TEMP;

end BUILD_CASE_ALT_0_MORE;

-------------------------------------------------------------------------------
-- Function to build individual case statement alternative template

function BUILD_CASE_STMT_ALT (FIRST,
                  SECOND,
                  THIRD    : ST_LIST_NODE_PTR;
                  OTHERS_CHK : BOOLEAN) return ST_LIST_NODE_PTR is

TEMP_4      : ST_LIST_NODE_PTR;
TEMP_6      : ST_LIST_NODE_PTR;
TEMP_7      : ST_LIST_NODE_PTR;

TEMP_8      : ST_LIST_NODE_PTR;
TEMP_10     : ST_LIST_NODE_PTR;
TEMP_11     : ST_LIST_NODE_PTR;
TEMP_12     : ST_LIST_NODE_PTR;
TEMP_13     : ST_LIST_NODE_PTR;
TEMP_14     : ST_LIST_NODE_PTR;

TEMP_4_STR    : STRING(1..27) := "Others clause caused fault ";
TEMP_6_STR    : STRING(1..24) := "No other condition true ";
TEMP_7_STR    : STRING(1..35) := "Body of others clause caused fault ";

TEMP_8_STR    : STRING(1..33) := "Current alternative caused fault ";
TEMP_10_STR   : STRING(1..25) := "Choice(s) condition true ";
TEMP_11_STR   : STRING(1..38) := "Current alternative body caused fault ";
TEMP_12_STR   : STRING(1..36) := "Current/previous choice cased fault ";
TEMP_13_STR   : STRING(1..28) := "Current choice caused fault ";
TEMP_14_STR   : STRING(1..29) := "Previous choice caused fault ";


begin -- BUILD CASE STMT ALT

-- Condition to check if alternative is an others alternative
if NOT OTHERS_CHK then

 -- Building prev choice node
 TEMP_14                    := new ST_LIST_NODE;
 TEMP_14.ROOT_FAULT(TEMP_14_STR'range) := TEMP_14_STR;
 if SECOND /= NULL then
```

```
 TEMP_14.NUMBER_OF_CHILDREN        := 1;
 TEMP_14.CHILDREN_NODES            := SECOND;
end if;

-- Building current choice node
TEMP_13                    := new ST_LIST_NODE;
TEMP_13.ROOT_FAULT(TEMP_13_STR'range) := TEMP_13_STR;
TEMP_13.NUMBER_OF_CHILDREN        := 1;
TEMP_13.CHILDREN_NODES            := FIRST;

-- Building current previous choice
TEMP_12                    := new ST_LIST_NODE;
TEMP_12.ROOT_FAULT(TEMP_12_STR'range) := TEMP_12_STR;
TEMP_12.TYPE_GATE                 := OR_GATE;
TEMP_12.NUMBER_OF_CHILDREN        := 2;
TEMP_12.CHILDREN_NODES            := TEMP_13;
TEMP_12.CHILDREN_NODES.PREV_ST_PTR   := TEMP_14;

-- Building condition true node
TEMP_10                    := new ST_LIST_NODE;
TEMP_10.ROOT_FAULT(TEMP_10_STR'range) := TEMP_10_STR;
TEMP_10.NUMBER_OF_CHILDREN        := 1;
TEMP_10.CHILDREN_NODES            := TEMP_12;

-- Building current alternative body node
TEMP_11                    := new ST_LIST_NODE;
TEMP_11.ROOT_FAULT(TEMP_11_STR'range) := TEMP_11_STR;
TEMP_11.NUMBER_OF_CHILDREN        := 1;
TEMP_11.CHILDREN_NODES            := THIRD;

-- Building current alternative node
TEMP_8                     := new ST_LIST_NODE;
TEMP_8.ROOT_FAULT(TEMP_8_STR'range)  := TEMP_8_STR;
TEMP_8.TYPE_GATE                  := AND_GATE;
TEMP_8.NUMBER_OF_CHILDREN         := 2;
TEMP_8.CHILDREN_NODES             := TEMP_10;
TEMP_8.CHILDREN_NODES.PREV_ST_PTR    := TEMP_11;

RETURN TEMP_8;

else

-- Building no other condition true node
TEMP_6                     := new ST_LIST_NODE;
TEMP_6.ROOT_FAULT(TEMP_6_STR'range)  := TEMP_6_STR;

-- Building body of other node
TEMP_7                     := new ST_LIST_NODE;
TEMP_7.ROOT_FAULT(TEMP_7_STR'range)  := TEMP_7_STR;
TEMP_7.NUMBER_OF_CHILDREN         := 1;
TEMP_7.CHILDREN_NODES             := THIRD;
```

```
  -- Building other node
  TEMP_4                        := new ST_LIST_NODE;
  TEMP_4.ROOT_FAULT(TEMP_4_STR'range)  := TEMP_4_STR;
  TEMP_4.TYPE_GATE              := AND_GATE;
  TEMP_4.NUMBER_OF_CHILDREN     := 2;
  TEMP_4.CHILDREN_NODES         := TEMP_6;
  TEMP_4.CHILDREN_NODES.PREV_ST_PTR   := TEMP_7;

  RETURN TEMP_4;

 end if;

end BUILD_CASE_STMT_ALT;

-------------------------------------------------------------------------------

-- Function to build or choice template for case statement

function BUILD_OR_CHOICE (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP_15      : ST_LIST_NODE_PTR;
 TEMP_16      : ST_LIST_NODE_PTR;
 TEMP_17      : ST_LIST_NODE_PTR;

 TEMP_15_STR   : STRING(1..36) := "Current/previous choice cased fault ";
 TEMP_16_STR   : STRING(1..28) := "Current choice caused fault ";
 TEMP_17_STR   : STRING(1..29) := "Previous choice caused fault ";

begin -- BUILD OR CHOICE

  -- Building current choice node
  TEMP_16                       := new ST_LIST_NODE;
  TEMP_16.ROOT_FAULT(TEMP_16_STR'range) := TEMP_16_STR;
  TEMP_16.NUMBER_OF_CHILDREN    := 1;
  TEMP_16.CHILDREN_NODES        := SECOND;

  -- Building prev choice node
  TEMP_17                       := new ST_LIST_NODE;
  TEMP_17.ROOT_FAULT(TEMP_17_STR'range) := TEMP_17_STR;
  if SECOND /= NULL then
   TEMP_17.NUMBER_OF_CHILDREN   := 1;
   TEMP_17.CHILDREN_NODES       := FIRST;
  end if;

  -- Building current previous choice
  TEMP_15                       := new ST_LIST_NODE;
  TEMP_15.ROOT_FAULT(TEMP_15_STR'range) := TEMP_15_STR;
  TEMP_15.TYPE_GATE             := OR_GATE;
  TEMP_15.NUMBER_OF_CHILDREN    := 2;
  TEMP_15.CHILDREN_NODES        := TEMP_16;
```

```
  TEMP_15.CHILDREN_NODES.PREV_ST_PTR  := TEMP_17;

  RETURN TEMP_15;

end BUILD_OR_CHOICE;

-----------------------------------------------------------------------

-- Function to build choice template for name rng_c

function BUILD_CHOICE_2 (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP       : ST_LIST_NODE_PTR;
 TEMP_STR   : STRING(1..35) := "Choice discreet range caused fault ";

begin -- Function to build discreet range node causing fault

 TEMP                    := new ST_LIST_NODE;
 TEMP.ROOT_FAULT(TEMP_STR'range) := TEMP_STR;
 TEMP.CHILDREN_NODES         := FIRST;
 TEMP.CHILDREN_NODES.PREV_ST_PTR := SECOND;
 TEMP.NUMBER_OF_CHILDREN     := 2;
 TEMP.TYPE_GATE              := OR_GATE;

 RETURN TEMP;

end BUILD_CHOICE_2;

-----------------------------------------------------------------------

-- Function to build choice statement template for sim expr .. sim expr

function BUILD_CHOICE_3 (FIRST, SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP       : ST_LIST_NODE_PTR;
 TEMP_STR   : STRING(1..36) := "Choice range specified caused fault ";

begin -- Beginning of function to build range specified node

 TEMP                    := new ST_LIST_NODE;
 TEMP.ROOT_FAULT(TEMP_STR'range) := TEMP_STR;
 TEMP.CHILDREN_NODES         := FIRST;
 TEMP.CHILDREN_NODES.PREV_ST_PTR := SECOND;
 TEMP.NUMBER_OF_CHILDREN     := 2;
 TEMP.TYPE_GATE              := OR_GATE;

 RETURN TEMP;

end BUILD_CHOICE_3;

-----------------------------------------------------------------------
```

```
-- Function to build procedure call template

function BUILD_PROC_CALL (FIRST   : ST_LIST_NODE_PTR;
                          TABLE   : SYS_TABLE;
                          COUNTER : INTEGER)
 return ST_LIST_NODE_PTR is

 TEMP          : ST_LIST_NODE_PTR;
 TEMP_2        : ST_LIST_NODE_PTR;
 TEMP_3        : ST_LIST_NODE_PTR;
 TEMP_4        : ST_LIST_NODE_PTR;
 TEMP_NF       : ST_LIST_NODE_PTR;

 TEMPORARY     : ST_LIST_NODE_PTR;

 TEMP_STR      : STRING(1..28) := "Procedure call caused fault ";
 TEMP_2_STR    : STRING(1..35) := "Procedure elaboration caused fault ";
 TEMP_3_STR    : STRING(1..28) := "Procedure body caused fault ";
 TEMP_4_STR    : STRING(1..24) := "Parameters caused fault ";
 TEMP_NF_STR   : STRING(1..29) := "Procedure not found on table ";

 NO_PARA       : BOOLEAN;

 PROCEDURE_NAME  : STRING(1..MAX_CHAR_LONG) := (OTHERS => ' ');
 FOUND         : BOOLEAN          := FALSE;
 LOCATION      : INTEGER;

begin -- function build procedure call

 -- Building of elaboration node
 TEMP_2                    := new ST_LIST_NODE;
 TEMP_2.ROOT_FAULT(TEMP_2_STR'range) := TEMP_2_STR;

 -- Building of procedure body node
 TEMP_3                    := new ST_LIST_NODE;
 TEMP_3.ROOT_FAULT(TEMP_3_STR'range) := TEMP_3_STR;
 TEMP_3.NUMBER_OF_CHILDREN       := 1;

 -- Conditional determines if sim_n or index_cmpon was used
 if FIRST.CHILDREN_NODES = NULL then
  NO_PARA                  := TRUE;
  TEMP_3.CHILDREN_NODES           := FIRST;
  TEMP_3.CHILDREN_NODES.NUMBER_OF_CHILDREN := 1;
 else
  NO_PARA                  := FALSE;
  TEMPORARY                  := new ST_LIST_NODE;
  TEMPORARY.ROOT_FAULT            := FIRST.CHILDREN_NODES.ROOT_FAULT;
  TEMP_3.CHILDREN_NODES           := TEMPORARY;
  TEMP_3.CHILDREN_NODES.PREV_ST_PTR    := NULL;
  TEMP_3.CHILDREN_NODES.NUMBER_OF_CHILDREN := 1;
 end if;
```

```
-- Identifies the name of the procedure
PROCEDURE_NAME := RETURN_ROOT_FAULT(TEMP_3.CHILDREN_NODES);

-- For loop to search table to see if procedure is in the table
for INDEX in 1..COUNTER loop
 if PROCEDURE_NAME = TABLE(INDEX).P_F_NAME then
  FOUND    := TRUE;
  LOCATION := INDEX;
 end if;
end loop;

-- Conditional will add sequence of statements to procedure if found
if FOUND then
 TEMP_3.CHILDREN_NODES.CHILDREN_NODES := TABLE(LOCATION).STMT_PTR;
else
 TEMP_NF                    := new ST_LIST_NODE;
 TEMP_NF.ROOT_FAULT(TEMP_NF_STR'range) := TEMP_NF_STR;
 TEMP_3.CHILDREN_NODES.CHILDREN_NODES := TEMP_NF;
end if;

-- Builds procedure call node
TEMP                    := new ST_LIST_NODE;
TEMP.ROOT_FAULT(TEMP_STR'range)  := TEMP_STR;
TEMP.TYPE_GATE          := OR_GATE;
TEMP.CHILDREN_NODES     := TEMP_2;
TEMP.CHILDREN_NODES.PREV_ST_PTR  := TEMP_3;

-- Will build parameters node if parameters used
if NOT NO_PARA then
 -- Builds the parameters node and attaches child
 TEMP_4                 := new ST_LIST_NODE;
 TEMP_4.ROOT_FAULT(TEMP_4_STR'range) := TEMP_4_STR;
 TEMP_4.NUMBER_OF_CHILDREN    := 1;
 TEMP_4.CHILDREN_NODES        := FIRST.CHILDREN_NODES.PREV_ST_PTR;

 TEMP.CHILDREN_NODES.PREV_ST_PTR.PREV_ST_PTR := TEMP_4;
 TEMP.NUMBER_OF_CHILDREN          := 3;
else
 TEMP.NUMBER_OF_CHILDREN          := 2;
end if;


 RETURN TEMP;

end BUILD_PROC_CALL;

-------------------------------------------------------------------------------

-- Function to build block statements template

function BUILD_BLOCK_STMT (FIRST  : ST_LIST_NODE_PTR;
```

```
                SECOND : ST_LIST_NODE_PTR;
                THIRD  : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP_1     : ST_LIST_NODE_PTR;
 TEMP_2     : ST_LIST_NODE_PTR;
 TEMP_3     : ST_LIST_NODE_PTR;
 TEMP_4     : ST_LIST_NODE_PTR;

 TEMP_1_STR  : STRING(1..29) := "Block Statement Caused Fault ";
 TEMP_2_STR  : STRING(1..23) := "Exception Caused Fault ";
 TEMP_3_STR  : STRING(1..24) := "Block Body Caused Fault ";
 TEMP_4_STR: STRING(1..34):= "Exception Propagated Caused Fault ";

begin

 -- Building block body node
 TEMP_3                    := new  ST_LIST_NODE;
 TEMP_3.ROOT_FAULT(TEMP_3_STR'range) := TEMP_3_STR;
 TEMP_3.NUMBER_OF_CHILDREN      := 1;
 TEMP_3.CHILDREN_NODES          := SECOND;

 -- Building Exception Node
 TEMP_2                    := new ST_LIST_NODE;
 TEMP_2.ROOT_FAULT(TEMP_2_STR'range) := TEMP_2_STR;
 TEMP_2.NUMBER_OF_CHILDREN      := 1;

 -- Conditional to determine if exception declared in block
 if THIRD /= NULL then
  TEMP_2.CHILDREN_NODES         := THIRD;
 else
  TEMP_4                    := new ST_LIST_NODE;
  TEMP_4.ROOT_FAULT(TEMP_4_STR'range) := TEMP_4_STR;
  TEMP_2.CHILDREN_NODES         := TEMP_4;
 end if;

 -- Building head node for template
 TEMP_1                    := new ST_LIST_NODE;
 TEMP_1.ROOT_FAULT(TEMP_1_STR'range) := TEMP_1_STR;

 -- Conditional to see if block named
 if FIRST /= NULL then
  -- Assigns name as child of block
  TEMP_1.NUMBER_OF_CHILDREN := 1;
  TEMP_1.CHILDREN_NODES     := FIRST;

  -- Assigns children to name block
  FIRST.TYPE_GATE           := OR_GATE;
  FIRST.NUMBER_OF_CHILDREN      := 2;
  FIRST.CHILDREN_NODES          := TEMP_2;
  FIRST.CHILDREN_NODES.PREV_ST_PTR := TEMP_3;
```

```
 else

  -- No name assigned to block
  TEMP_1.NUMBER_OF_CHILDREN       := 2;
  TEMP_1.TYPE_GATE                := OR_GATE;
  TEMP_1.CHILDREN_NODES           := TEMP_2;
  TEMP_1.CHILDREN_NODES.PREV_ST_PTR := TEMP_3;
 end if;

 return TEMP_1;

end BUILD_BLOCK_STMT;

-------------------------------------------------------------------------------

-- Function to Join list of exceptions

function JOIN_EXCEPTIONS (FIRST  : ST_LIST_NODE_PTR;
                          SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

begin

 if FIRST /= NULL then
  SECOND.CHILDREN_NODES.PREV_ST_PTR.CHILDREN_NODES  := FIRST;
  SECOND.CHILDREN_NODES.PREV_ST_PTR.NUMBER_OF_CHILDREN:= 1;
 end if;

 return SECOND;

end JOIN_EXCEPTIONS;

-------------------------------------------------------------------------------

-- Function to build template for one exception
-- Assumes 1 exception per alternative and simple not concat name

function BUILD_EXCEPTION (FIRST  : ST_LIST_NODE_PTR;
                          SECOND : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP    : ST_LIST_NODE_PTR;
 TEMP_B  : ST_LIST_NODE_PTR;

 TEMP_STR  : STRING(1..13) := "Exception(s) ";
 TEMP_B_STR : STRING(1..19) := "Other Exception(s) ";

begin

 -- Building other exceptions node
 TEMP_B                    := new ST_LIST_NODE;
 TEMP_B.ROOT_FAULT(TEMP_B_STR'range) := TEMP_B_STR;
```

```
-- Building Exceptions node
TEMP                        := new ST_LIST_NODE;
TEMP.ROOT_FAULT(TEMP_STR'range)    := TEMP_STR;
TEMP.TYPE_GATE              := OR_GATE;
TEMP.NUMBER_OF_CHILDREN     := 2;
TEMP.CHILDREN_NODES         := FIRST;
TEMP.CHILDREN_NODES.CHILDREN_NODES  := SECOND;
TEMP.CHILDREN_NODES.PREV_ST_PTR    := TEMP_B;

RETURN TEMP;

end BUILD_EXCEPTION;

-------------------------------------------------------------------------------
-- Function to build function call template if required

function CHECK_FUNCTION (FIRST   : ST_LIST_NODE_PTR;
                         TABLE   : SYS_TABLE;
                         COUNTER : INTEGER)
return  ST_LIST_NODE_PTR is

TEMP   : ST_LIST_NODE_PTR;
TEMP_2 : ST_LIST_NODE_PTR;
TEMP_3 : ST_LIST_NODE_PTR;
TEMP_4 : ST_LIST_NODE_PTR;

TEMP_STR   : STRING(1..27) := "Function Call Caused Fault ";
TEMP_2_STR :STRING(1..34)  := "Function Elaboration Caused Fault ";
TEMP_3_STR : STRING(1..27) := "Function Body Caused Fault ";
TEMP_4_STR : STRING(1..24) := "Parameters Caused Fault ";

FUNCTION_NAME : STRING(1..MAX_CHAR_LONG) := (OTHERS => ' ');
FOUND      : BOOLEAN        := FALSE;
NO_PARA    : BOOLEAN        := FALSE;

LOCATION   : INTEGER;

begin -- Function Check Function

-- Determines location of name (Simple name or index component)
if FIRST.CHILDREN_NODES = NULL then
 NO_PARA := TRUE; -- Simple name used
end if;

-- Returns the name of the function, if it is a function
if NO_PARA then
 FUNCTION_NAME := RETURN_ROOT_FAULT(FIRST);
else
 FUNCTION_NAME := RETURN_ROOT_FAULT(FIRST.CHILDREN_NODES);
end if;
```

```
-- Determines if it is on the Procedure/Function Table
for INDEX in  1..COUNTER loop
 if FUNCTION_NAME = TABLE(INDEX).P_F_NAME then
  FOUND    := TRUE;
  LOCATION  := INDEX;
 end if;
end loop;

-- If not on table then return pointer sent in, if found then
-- return Function Call Template

if NOT FOUND then
 -- Name not found on table therefore return original node
 return FIRST;
ELSE
 TEMP_2                    := new ST_LIST_NODE;
 TEMP_2.ROOT_FAULT(TEMP_2_STR'range) := TEMP_2_STR;

 TEMP_3                    := new ST_LIST_NODE;
 TEMP_3.ROOT_FAULT(TEMP_3_STR'range) := TEMP_3_STR;
 TEMP_3.NUMBER_OF_CHILDREN        := 1;

 -- The name of the function
 if NO_PARA then
  TEMP_3.CHILDREN_NODES          := FIRST;
 else
  TEMP_3.CHILDREN_NODES          := FIRST.CHILDREN_NODES;
 end if;

 TEMP_3.CHILDREN_NODES.NUMBER_OF_CHILDREN := 1;

 -- The sequence of statements for the name
 TEMP_3.CHILDREN_NODES.CHILDREN_NODES := TABLE(LOCATION).STMT_PTR;

 -- Builds function call node
 TEMP                    := new ST_LIST_NODE;
 TEMP.ROOT_FAULT(TEMP_STR'range) := TEMP_STR;
 TEMP.TYPE_GATE             := OR_GATE;
 TEMP.CHILDREN_NODES          := TEMP_2;
 TEMP.CHILDREN_NODES.PREV_ST_PTR := TEMP_3;

 -- Conditional to build parameters node if needed
 if NOT NO_PARA then
  -- Builds the parameters node and attaches child
  TEMP_4                    := new ST_LIST_NODE;
  TEMP_4.ROOT_FAULT(TEMP_4_STR'range) := TEMP_4_STR;
  TEMP_4.NUMBER_OF_CHILDREN        := 1;
  TEMP_4.CHILDREN_NODES          := FIRST.CHILDREN_NODES.PREV_ST_PTR;

  FIRST.CHILDREN_NODES.PREV_ST_PTR   := NULL;
```

```
   TEMP.CHILDREN_NODES.PREV_ST_PTR.PREV_ST_PTR := TEMP_4;
   TEMP.NUMBER_OF_CHILDREN                     := 3;
  else
   TEMP.NUMBER_OF_CHILDREN := 2;
  end if;

  return TEMP;

 end if;

end CHECK_FUNCTION;

-------------------------------------------------------------------------------
-- Function to build raise statement template

function BUILD_RAISE (FIRST : ST_LIST_NODE_PTR)
 return ST_LIST_NODE_PTR is

 TEMP     : ST_LIST_NODE_PTR;
 TEMP_2   : ST_LIST_NODE_PTR;
 TEMP_3   : ST_LIST_NODE_PTR;
 TEMP_4   : ST_LIST_NODE_PTR;

 TEMP_STR  : STRING(1..29) := "Raise Statement Caused Fault ";
 TEMP_2_STR : STRING(1..36) := "Wrong Exception Raised Caused Fault ";
 TEMP_3_STR : STRING(1..31) := "Exception Handler Caused Fault ";
 TEMP_4_STR : STRING(1..33) := "Exception not defined in program ";

begin

 TEMP_2                      := new ST_LIST_NODE;
 TEMP_2.ROOT_FAULT(TEMP_2_STR'range) := TEMP_2_STR;

 TEMP_3                      := new ST_LIST_NODE;
 TEMP_3.ROOT_FAULT(TEMP_3_STR'range) := TEMP_3_STR;

 -- Conditional to determine if exception stated
 if FIRST /= NULL then
  TEMP_3.NUMBER_OF_CHILDREN := 1;
  TEMP_3.CHILDREN_NODES    := FIRST;
 else
  TEMP_4                      := new ST_LIST_NODE;
  TEMP_4.ROOT_FAULT(TEMP_4_STR'range) := TEMP_4_STR;

  TEMP_3.NUMBER_OF_CHILDREN := 1;
  TEMP_3.CHILDREN_NODES    := TEMP_4;
 end if;

 TEMP                      := new ST_LIST_NODE;
 TEMP.ROOT_FAULT(TEMP_STR'range)   := TEMP_STR;
```

```
 TEMP.TYPE_GATE                := OR_GATE;
 TEMP.NUMBER_OF_CHILDREN       := 2;
 TEMP.CHILDREN_NODES           := TEMP_2;
 TEMP.CHILDREN_NODES.PREV_ST_PTR  := TEMP_3;

 return TEMP;

end BUILD_RAISE;
-------------------------------------------------------------------------------
-- Function to see if exception in exception table

function CHECK_EXCEPT (FIRST          : ST_LIST_NODE_PTR;
              EX_TAB          : EXCEPT_TABLE;
              EXCEPT_COUNTER  : INTEGER)
 return ST_LIST_NODE_PTR is

 TEMP         : ST_LIST_NODE_PTR;
 TEMP_STR     : STRING(1..23)        := "Exception not on table ";

 NEW_PTR      : ST_LIST_NODE_PTR     := new ST_LIST_NODE;

 EXCEPTION_NAME : STRING(1..MAX_CHAR_LONG) := (OTHERS => ' ');

 FOUND        : BOOLEAN              := FALSE;
 LOCATION     : INTEGER;

begin

 -- Conditional to determine if exception name stated
 if FIRST = NULL then
  return FIRST;
 else
  -- Assigns name of exception to variable
  EXCEPTION_NAME := RETURN_ROOT_FAULT(FIRST);

  -- Loop will determine if exception is on table
  for INDEX in  1..EXCEPT_COUNTER loop
   if EXCEPTION_NAME = EX_TAB(INDEX).EXCEPT_NAME then
    FOUND    := TRUE;
    LOCATION := INDEX;
   end if;
  end loop;

  if FOUND then
   -- Option will attach sequence of statements for exception
   NEW_PTR              := EX_TAB(LOCATION).EXCEPT_PTR;
   FIRST.CHILDREN_NODES    := NEW_PTR;
   FIRST.NUMBER_OF_CHILDREN := 1;
  else
   -- Option will create node stating exception not defined
   TEMP                 := new ST_LIST_NODE;
```

```
    TEMP.ROOT_FAULT(TEMP_STR'range) := TEMP_STR;

    FIRST.CHILDREN_NODES          := TEMP;
    FIRST.NUMBER_OF_CHILDREN      := 1;
   end if;

   return FIRST;

  end if;

end CHECK_EXCEPT;

-------------------------------------------------------------------------------

end FAULT_TREE_GENERATOR;
```

C. PACKAGE SPECIFICATION FOR TRAVERSE PACKAGE

```
with TEXT_IO, FAULT_TREE_GENERATOR;
use  TEXT_IO, FAULT_TREE_GENERATOR;

package TRAVERSE_PKG is

package INTEGER_INOUT is new INTEGER_IO(INTEGER);
use   INTEGER_INOUT;

package BOOLEAN_INOUT is new ENUMERATION_IO(BOOLEAN);
use   BOOLEAN_INOUT;

 MAX_NODES : INTEGER := 1000;  -- Arbitrary number

 subtype NODE_NUMBER  is INTEGER  range 0..MAX_NODES;
 subtype LEVEL_NUMBER is INTEGER  range 0..MAX_NODES;
 subtype QUEUE_LIST   is INTEGER  range 0..MAX_NODES;

 -- Declaration of variables for Queue
 type QUEUE_RECORD is
  record
   NODE    : ST_LIST_NODE_PTR;
   NODE_NO  : NODE_NUMBER;
   LEVEL_NO : LEVEL_NUMBER;
  end record;

 type QUEUE is array (1..MAX_NODES) of QUEUE_RECORD;

 TREE     : QUEUE;
 FRONT    : QUEUE_LIST := 0;
 REAR     : QUEUE_LIST := 0;

 PARENT   : ST_LIST_NODE_PTR;
 CHILD    : ST_LIST_NODE_PTR;
 EMPTY_QUE : BOOLEAN := FALSE;

 PARENT_NO : NODE_NUMBER  := 0;
 CHILD_NO  : NODE_NUMBER  := 0;
 COUNTER   : NODE_NUMBER  := 0;

 GATE     : STRING(1..5) := (others => ' ');
 GATE_VALUE : GATE_TYPE   := 0;

 -- Declaration of variables for stack
 type STACK_RECORD is
  record
   NODE   : ST_LIST_NODE_PTR;
   NODE_NO : NODE_NUMBER;
```

```
  end record;

 type STACK is array (1..MAX_NODES) of STACK_RECORD;

 TEMP_STACK : STACK;
 MAIN_STACK : STACK;

 MAXSTK   : INTEGER := MAX_NODES;
 TOP      : INTEGER := 0;
 T_TOP    : INTEGER := 0;

function IS_EMPTY (FRONT : QUEUE_LIST) return boolean;

procedure ENQUE (TTREE      : in out QUEUE;
          FRONT     : in out QUEUE_LIST;
          REAR      : in out QUEUE_LIST;
          POINTER   : in out ST_LIST_NODE_PTR;
          NUMBER    : in out NODE_NUMBER;
          LEV_NO    : in out LEVEL_NUMBER);

procedure DEQUE (TTREE      : in out QUEUE;
          FRONT     : in out QUEUE_LIST;
          REAR      : in out QUEUE_LIST;
          POINTER   :    out ST_LIST_NODE_PTR;
          NUMBER    :    out NODE_NUMBER;
          LEV_NO    :    out LEVEL_NUMBER);

procedure TREE_TRAVERSAL (ROOT  : ST_LIST_NODE_PTR);

procedure FTE_FILE1 (ROOT    : ST_LIST_NODE_PTR);

function EMPTY(TOP : INTEGER) return boolean;

procedure PUSH (T_STACK   : in out STACK;
          TOP_NO    : in out INTEGER;
          NODE      : in out ST_LIST_NODE_PTR);

procedure POP (T_STACK    : in out STACK;
          TOP_NO    : in out INTEGER;
          NODE      :    out ST_LIST_NODE_PTR);

procedure CREATE_FILE  (ROOT   : ST_LIST_NODE_PTR;
                FTE_FILE : FILE_TYPE);

end TRAVERSE_PKG;
```

D. PACKAGE BODY FOR TRAVERSE PACKAGE

```
package body TRAVERSE_PKG is

-------------------------------------------------------------------------------

 function IS_EMPTY (FRONT : QUEUE_LIST) return boolean is

 begin -- function is_empty

  if FRONT = 0 then
   return true;
  else
   return false;
  end if;

 end IS_EMPTY;

-------------------------------------------------------------------------------

procedure ENQUE (TTREE    : in out QUEUE;
          FRONT    : in out QUEUE_LIST;
          REAR     : in out QUEUE_LIST;
          POINTER  : in out ST_LIST_NODE_PTR;
          NUMBER   : in out NODE_NUMBER;
          LEV_NO   : in out LEVEL_NUMBER) is

 begin -- procedure enque

  if FRONT = 1 and REAR = MAX_NODES then
   new_line;
   put_line("OVERFLOW ERROR. ");
   new_line;
  else
   if FRONT = 0 then          -- Queue empty
    FRONT := 1;
    REAR  := 1;
   elsif REAR = MAX_NODES then  -- At end of queue
    REAR  := 1;
   else                  -- Increment queue
    REAR  := REAR + 1;
   end if;
   TTREE(REAR).NODE    := POINTER;
   TTREE(REAR).NODE_NO := NUMBER;
   TTREE(REAR).LEVEL_NO := LEV_NO;
  end if;

 end ENQUE;

-------------------------------------------------------------------------------

procedure DEQUE (TTREE    : in out QUEUE;
          FRONT    : in out QUEUE_LIST;
```

```
        REAR     : in out QUEUE_LIST;
        POINTER  :  out ST_LIST_NODE_PTR;
        NUMBER   :  out NODE_NUMBER;
        LEV_NO   :  out LEVEL_NUMBER) is

begin -- procedure deque

 if FRONT = 0 then              -- Trying to deque null
  new_line;
  put_line("UNDERFLOW ERROR. ");
  new_line;
 else
  POINTER := TTREE(FRONT).NODE;
  NUMBER  := TTREE(FRONT).NODE_NO;
  LEV_NO  := TTREE(FRONT).LEVEL_NO;
  if FRONT = REAR then         -- 1 item in queue
   FRONT := 0;
   REAR  := 0;
  elsif FRONT = MAX_NODES then -- At end of queue
   FRONT := 1;
  else                         -- Anywhere in queue
   FRONT := FRONT + 1;
  end if; -- if front = rear

 end if; -- if front = 0

end DEQUE;
-------------------------------------------------------------------------------
procedure TREE_TRAVERSAL (ROOT : ST_LIST_NODE_PTR) is


 DUMMY_VARIABLE : INTEGER := 0;

begin -- procedure tree_traversal

 -- Initialization of Variables used
 FRONT       := 0;
 REAR        := 0;

 EMPTY_QUE   := FALSE;

 PARENT_NO   := 0;
 CHILD_NO    := 0;
 COUNTER     := 0;

 GATE_VALUE  := 0;

 -- Heading for traversal
 new_line;
 SET_COL(11);
 PUT("LITERAL TREE OUTPUT");
```

```
new_line;
SET_COL(11);
PUT ("-----------------------------------");
new_line(2);
SET_COL(17);
PUT ("Parent");
new_line;
PUT ("Parent");
SET_COL(11);
PUT ("Node ");
SET_COL(18);
PUT ("GATE ");
SET_COL(24);
PUT ("Fault for Node ");
new_line;
PUT ("------");
SET_COL(11);
PUT ("----");
SET_COL(18);
PUT ("----");
SET_COL(24);
PUT ("--------------");
new_line;

-- Setting Head of tree to the root
PARENT := ROOT;
ENQUE(TREE, FRONT, REAR, PARENT, PARENT_NO, DUMMY_VARIABLE);

-- Printing the values of the root
new_line;
SET_COL(7);
PUT (" => ");
PUT (PARENT_NO, width => 5);
SET_COL(18);
PUT ("Null ");
SET_COL(24);
PUT (RETURN_ROOT_FAULT(PARENT));
new_line;

-- Traversing the remaining nodes and printing the values
while not IS_EMPTY(FRONT) loop

  DEQUE(TREE, FRONT, REAR, PARENT, PARENT_NO, DUMMY_VARIABLE);

  if PARENT /= NULL then

    -- Sets pointer to first child
    CHILD := RETURN_CHILD(PARENT);

    -- Function will retrieve gate type of parent
    GATE_VALUE := RETURN_GATE_TYPE(PARENT);
```

```
    -- Conditional to convert type gate from integer to string
    if   GATE_VALUE = 0 then
      GATE := "Null ";
    elsif GATE_VALUE = 1 then
      GATE := "And  ";
    else
      GATE := "Or   ";
    end if;

    -- Prints children and puts children on queue
    if CHILD /= NULL then
      -- Conditional determines the child and puts the child on array
      while CHILD /= NULL loop
        COUNTER  := COUNTER + 1;
        CHILD_NO := COUNTER;
        new_line;
        PUT (PARENT_NO, width => 5);
        SET_COL(7);
        PUT (" => ");
        PUT (CHILD_NO,  width => 5);
        SET_COL(18);
        PUT (GATE);
        SET_COL(24);
        PUT (RETURN_ROOT_FAULT(CHILD));
        new_line;
        ENQUE(TREE, FRONT, REAR, CHILD, CHILD_NO, DUMMY_VARIABLE);
        CHILD := RETURN_CHILD_PREV(CHILD);
      end loop;  -- while child...

    end if;     -- if child...

  end if;       -- if parent...

end loop;       -- while not...


end TREE_TRAVERSAL;

-------------------------------------------------------------------------------

procedure FTE_FILE1  (ROOT : ST_LIST_NODE_PTR) is


 DUMMY_VARIABLE : INTEGER := 0;

begin -- procedure FTE_FILE

 -- Initialization of Variables used
 FRONT       := 0;
 REAR        := 0;
```

```
EMPTY_QUE     := FALSE;

PARENT_NO     := 0;
CHILD_NO      := 0;
COUNTER       := 0;

GATE_VALUE    := 0;

-- Heading for traversal
new_line;
SET_COL(11);
PUT ("F T E  F I L E  O U T P U T");
new_line;
SET_COL(11);
PUT ("---------------------------");
new_line;
new_line;
SET_COL(11);
PUT ("Node ");
SET_COL(24);
PUT ("Fault for Node ");
new_line;
SET_COL(11);
PUT ("----");
SET_COL(24);
PUT ("---------------");
new_line;

-- Setting Head of tree to the root
PARENT := ROOT;
ENQUE(TREE, FRONT, REAR, PARENT, PARENT_NO, DUMMY_VARIABLE);

-- Printing the values of the root
new_line;
SET_COL(11);
PUT (PARENT_NO, width => 5);
SET_COL(24);
PUT (RETURN_ROOT_FAULT(PARENT));
new_line;

-- Traversing the remaining nodes and printing the values
while not IS_EMPTY(FRONT) loop

  DEQUE(TREE, FRONT, REAR, PARENT, PARENT_NO, DUMMY_VARIABLE);

  if PARENT /= NULL then

    -- Sets pointer to first child
    CHILD := RETURN_CHILD(PARENT);
```

```
    -- Prints children and puts children on queue
    if CHILD /= NULL then
     -- Conditional determines the child and puts the child on array
     while CHILD /= NULL loop
      COUNTER := COUNTER + 1;
      CHILD_NO := COUNTER;
      new_line;
      SET_COL(11);
      PUT (CHILD_NO, width => 5);
      SET_COL(24);
      PUT (RETURN_ROOT_FAULT(CHILD));
      new_line;
      ENQUE(TREE, FRONT, REAR, CHILD, CHILD_NO, DUMMY_VARIABLE);
      CHILD := RETURN_CHILD_PREV(CHILD);
     end loop;  -- while child...

    end if;     -- if child...

   end if;      -- if parent...

 end loop;      -- while not...


end FTE_FILE1;

-------------------------------------------------------------------------------
function EMPTY(TOP : INTEGER) return boolean is

begin -- Function Empty

 if TOP = 0 then
  return TRUE;
 else
  return FALSE;
 end if;

end EMPTY;

-------------------------------------------------------------------------------
procedure PUSH (T_STACK   : in out STACK;
          TOP_NO    : in out INTEGER;
          NODE      : in out ST_LIST_NODE_PTR) is

begin

 if TOP_NO = MAXSTK then
  NEW_LINE;
  PUT_LINE("OVERFLOW. ERROR. ERROR.");
  NEW_LINE;
 else
  TOP_NO            := TOP_NO + 1;
```

```
  T_STACK(TOP_NO).NODE   := NODE;
 end if;

end PUSH;

-------------------------------------------------------------------------------

procedure POP (T_STACK    : in out STACK;
          TOP_NO     : in out INTEGER;
          NODE       :   out ST_LIST_NODE_PTR) is

begin

 if TOP_NO = 0 then
  NEW_LINE;
  PUT_LINE("UNDERFLOW. ERROR. ERROR");
  NEW_LINE;
 else
  NODE    := T_STACK(TOP_NO).NODE;
  TOP_NO  := TOP_NO - 1;
 end if;

end POP;

-------------------------------------------------------------------------------

procedure CREATE_FILE (ROOT    : ST_LIST_NODE_PTR;
               FILE_FILE : FILE_TYPE) is

 -- Declaration of local variables used in procedure
 B_LINE     : NATURAL  := 0;
 E_LINE     : NATURAL  := 0;
 X_COORD    : INTEGER  := 0;
 Y_COORD    : INTEGER  := 0;

 VARIABLE   : INTEGER  := 0;
 TOGGLE     : BOOLEAN        := FALSE;

 VARX       : constant INTEGER := 75;
 VARY       : constant INTEGER := 85;

 LEVEL_CHG  : INTEGER  := 0;
 Y_LEVEL_NO : INTEGER  := 0;

 PLEVEL_NO  : INTEGER  := 0;
 CLEVEL_NO  : INTEGER  := 0;
 OLD_PLEVEL : INTEGER  := 0;

 LAST_CHAR  : INTEGER  := 0;

 LABEL      : STRING(1..MAX_CHAR_SHORT) := (OTHERS => ' ');
 ROOT_FAULT : STRING(1..MAX_CHAR_LONG)  := (OTHERS => ' ');
 FILE_NAME  : STRING(1..MAX_CHAR_LONG)  := (OTHERS => ' ');
```

```
begin

 -- Initialization of variables declared in specification
 FRONT        := 0;
 REAR         := 0;

 EMPTY_QUE    := FALSE;

 PARENT_NO    := 0;
 CHILD_NO     := 0;
 COUNTER      := 0;

 GATE_VALUE   := 0;

 -- Setting Head of tree to the root
 PARENT := ROOT;
 ENQUE(TREE, FRONT, REAR, PARENT, PARENT_NO, PLEVEL_NO);

 PUSH_LABEL   (PARENT, INTEGER'IMAGE(PARENT_NO));
 PUSH_FILE_N  (PARENT, "EXAMPLE.A");
 PUSH_X_COORD  (PARENT, X_COORD);
 PUSH_Y_COORD  (PARENT, Y_COORD);


 -- Traversing the remaining nodes and printing the values
 while not IS_EMPTY(FRONT) loop

   DEQUE(TREE, FRONT, REAR, PARENT, PARENT_NO, PLEVEL_NO);

   if PARENT /= NULL then

    if PLEVEL_NO /= OLD_PLEVEL then
     X_COORD  := 0;
    end if;

    -- Sets pointer to first child
    CHILD := RETURN_CHILD(PARENT);


    -- Prints children and puts children on queue
    if CHILD /= NULL then
     -- Conditional determines the child and puts the child on array
     while CHILD /= NULL loop
      COUNTER  := COUNTER + 1;
      CHILD_NO := COUNTER;

      PUSH_LABEL  (CHILD, INTEGER'IMAGE(CHILD_NO));
      PUSH_FILE_N (CHILD, "EXAMPLE.A");

      PUSH_X_COORD(CHILD, X_COORD);
```

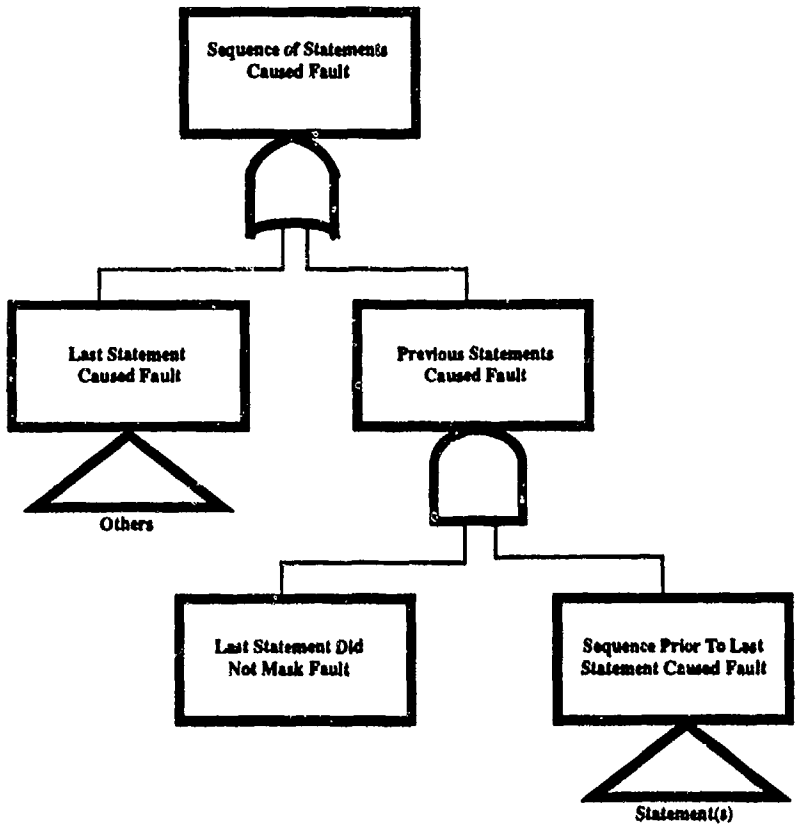
```
        x_coord := x_coord + varx;

        PUSH_Y_COORD(CHILD, RETURN_Y_COORD(PARENT) + VARY);

        CLEVEL_NO  := PLEVEL_NO + 1;

        ENQUE(TREE, FRONT, REAR, CHILD, CHILD_NO, CLEVEL_NO);
        CHILD := RETURN_CHILD_PREV(CHILD);

        if CHILD = NULL then
         OLD_PLEVEL := PLEVEL_NO;
        end if;

      end loop;  -- while child...

     end if;     -- if child...

   end if;       -- if parent...

end loop;        -- while not...

-- Initialization of variables
PARENT_NO   := 0;
CHILD_NO    := 0;
COUNTER     := 0;

-- Placing the root node into the stack
PARENT  := ROOT;
PUSH(MAIN_STACK, TOP, PARENT);

-- Conditional to do depth first traversal of tree
while NOT EMPTY(TOP) loop
 POP(MAIN_STACK, TOP, PARENT);

 -- Putting the label of node into the new file
 LABEL    := RETURN_LABEL(PARENT);
 LAST_CHAR := LABEL'LAST;

 while LABEL(LAST_CHAR) = ' ' and LAST_CHAR /= 0 loop
  LAST_CHAR := LAST_CHAR - 1;
 end loop;

 if LAST_CHAR = 0 then
  PUT(FTE_FILE, "");
  NEW_LINE(FTE_FILE);
 else
  PUT(FTE_FILE, LABEL(1..LAST_CHAR));
  NEW_LINE(FTE_FILE);
 end if;

-- Putting the fault of the node into the new file
```

```
ROOT_FAULT   := RETURN_ROOT_FAULT(PARENT);
LAST_CHAR    := ROOT_FAULT'LAST;

while ROOT_FAULT(LAST_CHAR) = ' ' and LAST_CHAR /= 0 loop
 LAST_CHAR := LAST_CHAR - 1;
end loop;

if LAST_CHAR = 0 then
 PUT(FTE_FILE, "");
 NEW_LINE(FTE_FILE);
else
 PUT(FTE_FILE, ROOT_FAULT(1..LAST_CHAR));
 NEW_LINE(FTE_FILE);
end if;

-- Putting file of node into the new file
FILE_NAME    := RETURN_FILE_NAME(PARENT);
LAST_CHAR    := FILE_NAME'LAST;

while FILE_NAME(LAST_CHAR) = ' ' and LAST_CHAR /= 0 loop
 LAST_CHAR := LAST_CHAR - 1;
end loop;

if LAST_CHAR = 0 then
 PUT(FTE_FILE, "");
 NEW_LINE(FTE_FILE);
else
 PUT(FTE_FILE, FILE_NAME(1..LAST_CHAR));
 NEW_LINE(FTE_FILE);
end if;

-- Putting line 4 of node(Line no's, X & Y coord, node, gate, children)
PUT(FTE_FILE, RETURN_START_L(PARENT),    width => 0);
PUT(FTE_FILE, ' ');
PUT(FTE_FILE, RETURN_END_L(PARENT),      width => 0);
PUT(FTE_FILE, ' ');
PUT(FTE_FILE, RETURN_X_COORD(PARENT),    width => 0);
PUT(FTE_FILE, ' ');
PUT(FTE_FILE, RETURN_Y_COORD(PARENT),    width => 0);
PUT(FTE_FILE, ' ');
PUT(FTE_FILE, RETURN_NODE_TYPE(PARENT),  width => 0);
PUT(FTE_FILE, ' ');
PUT(FTE_FILE, RETURN_GATE_TYPE(PARENT),  width => 0);
PUT(FTE_FILE, ' ');
PUT(FTE_FILE, RETURN_NO_CHILDREN(PARENT), width => 0);
NEW_LINE(FTE_FILE);

CHILD := RETURN_CHILD(PARENT);
if CHILD /= null then

 -- Conditional to determine children of parent
```

```
    while CHILD /= null loop
     COUNTER := COUNTER + 1;
     CHILD_NO := COUNTER;
     PUSH(TEMP_STACK, T_TOP, CHILD);
     CHILD   := RETURN_CHILD_PREV(CHILD);
    end loop;

    -- Conditional to place children on stack in order
    while not EMPTY(T_TOP) loop
     POP  (TEMP_STACK, T_TOP, CHILD);
     PUSH (MAIN_STACK, TOP,   CHILD);
    end loop;

   end if;

  end loop;

end CREATE_FILE;

-------------------------------------------------------------------------------

end TRAVERSE_PKG;
```

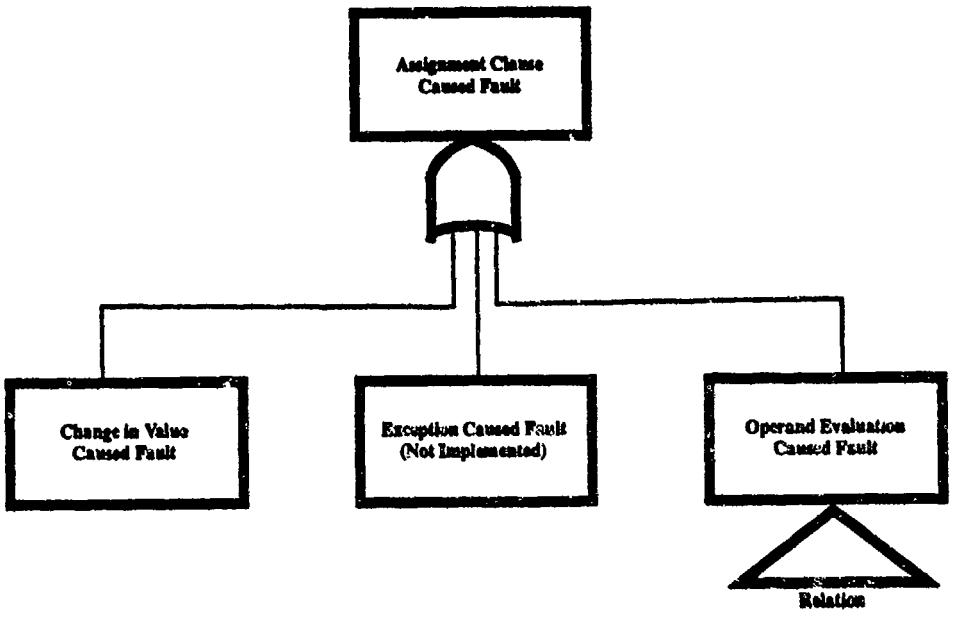
APPENDIX C: ADA STRUCTURE TEMPLATES

Sequence of Statements Template



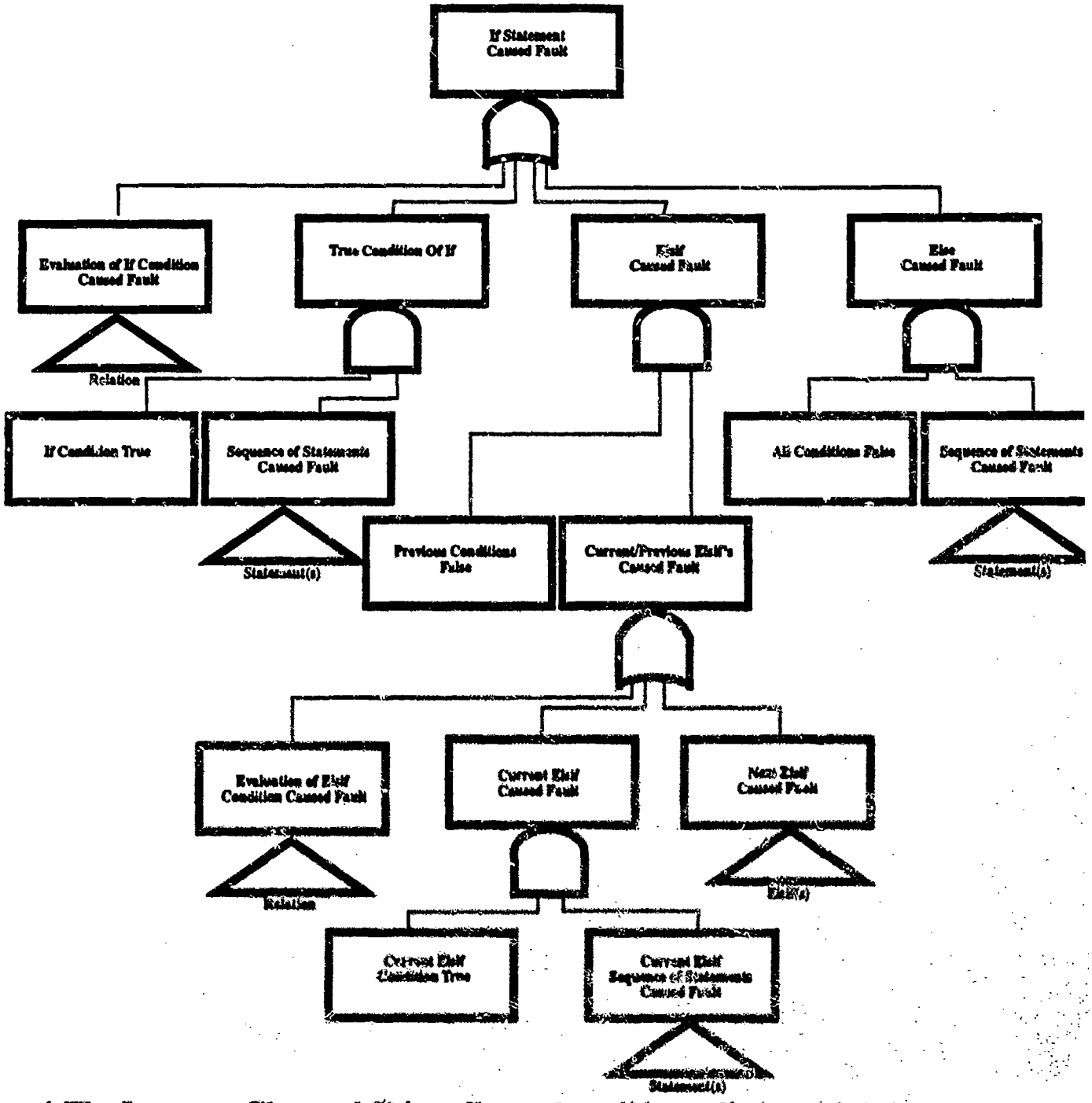
*** Sequence of statements template was not depicted in the works of Leveson, Cha, and Shimeall.**

Assignment Template



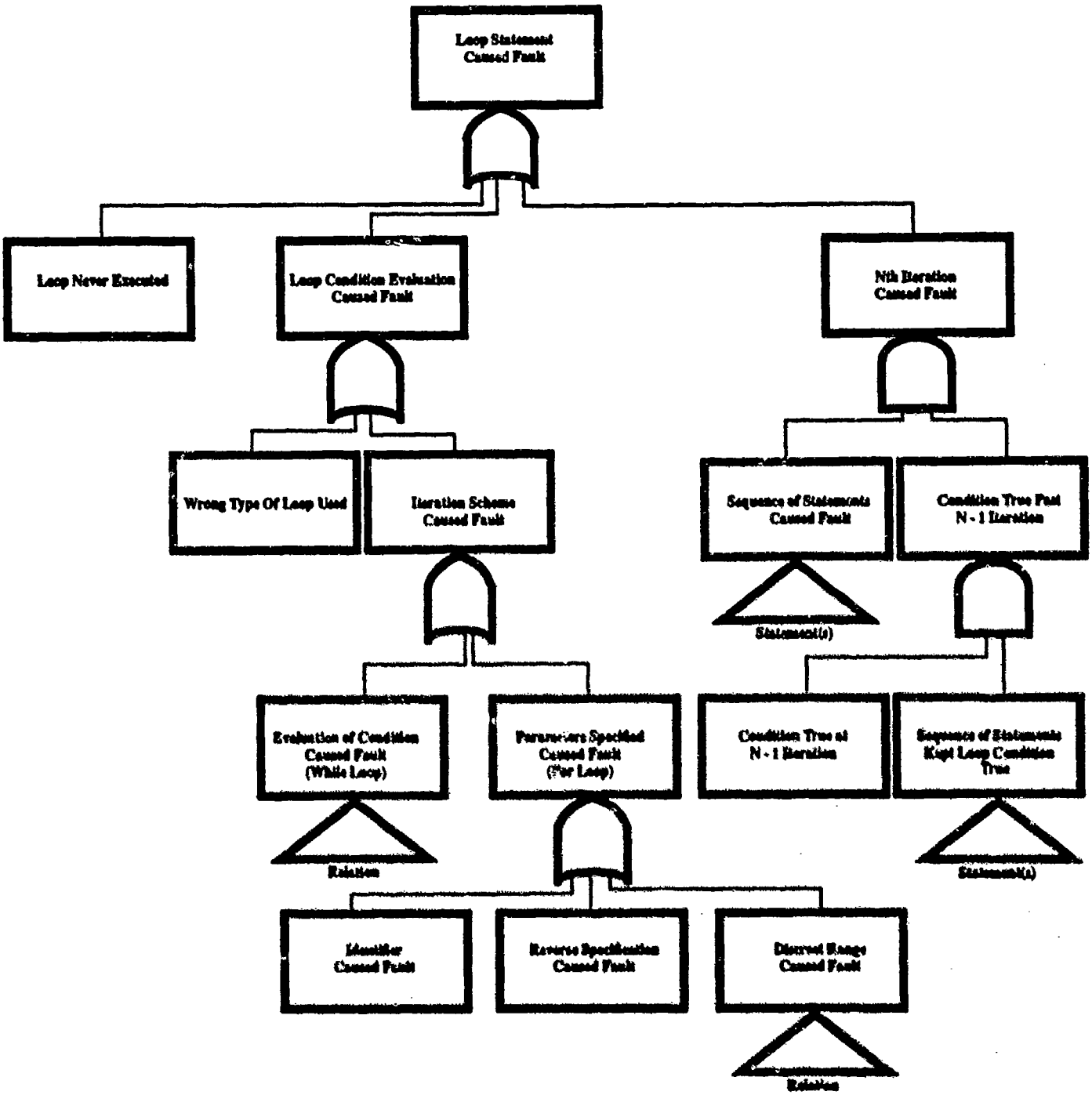
*** There is no difference between the tool generated Assignment Template and the Leveson, Cha, and Shimeall template.**

If Then Elsif Else Template



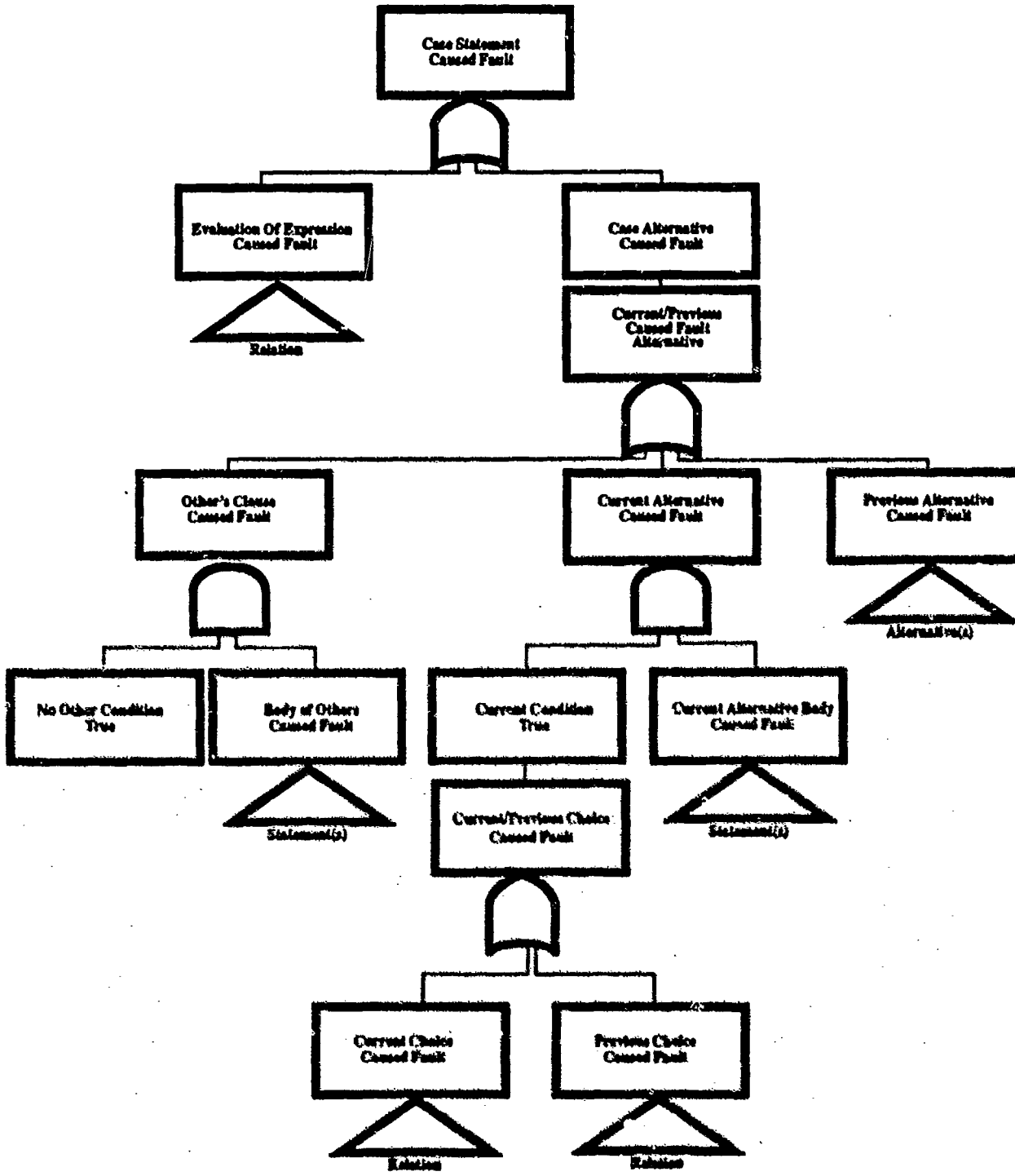
* The Leveson, Cha, and Shimeall template did not distinguish between the "if" associated nodes and the "elsif" associated nodes, but the tool generated template presented the two separately.

Loop Template



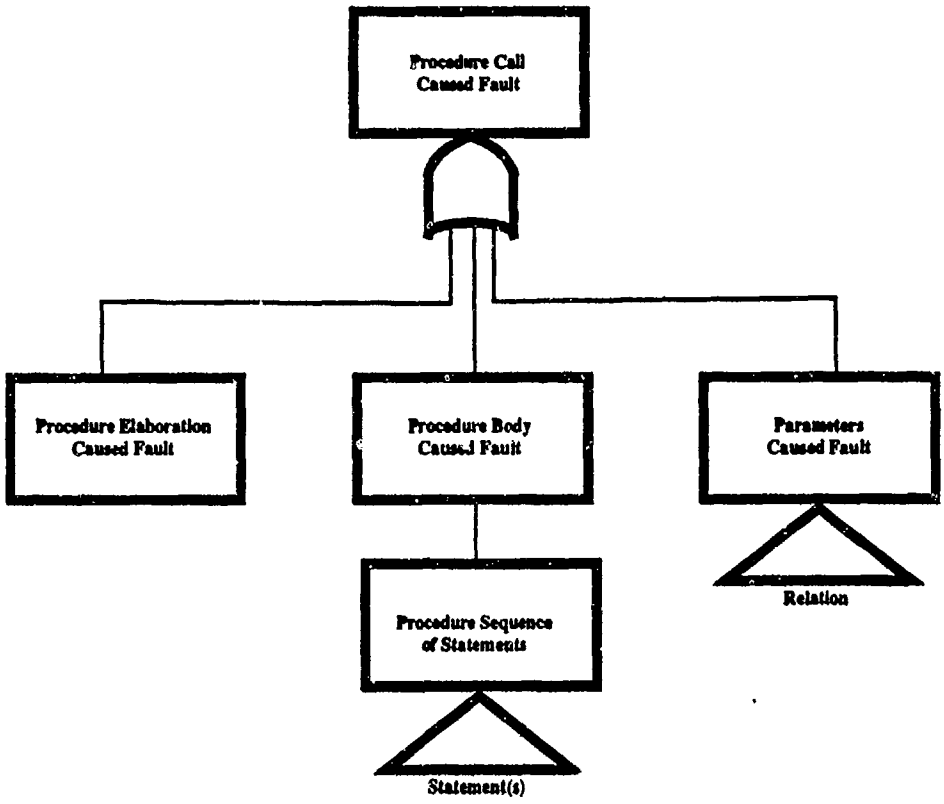
* The tool generated Loop Template distinguished between the different types of loops and expanded the associated "condition" nodes and the Leveson, Cha and Shimeall template did not.

Case Template



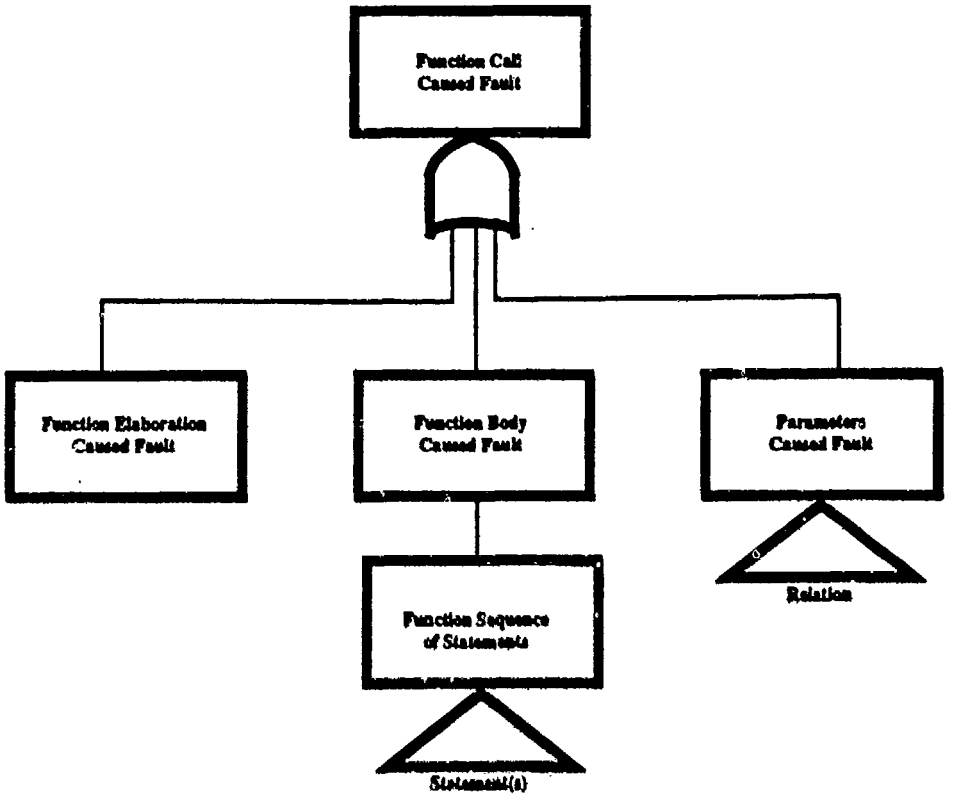
* The tool generated Case Template physically placed the "Other's Clause Caused Fault" and associated nodes on a lower level then the Leveson, Cha, and Shimeall template, nevertheless, the two templates are equivalent.

Procedure Call Template



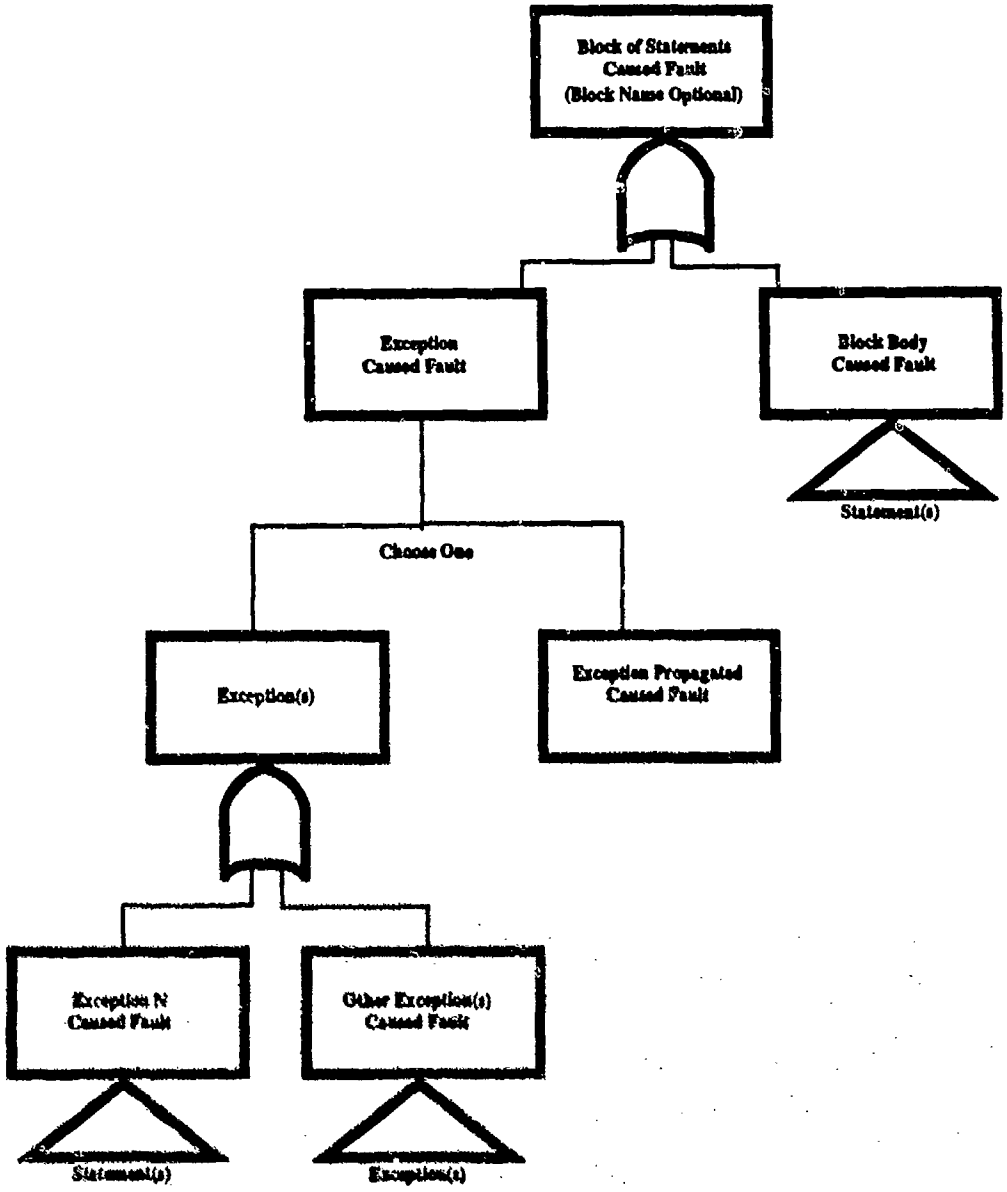
*** The tool generated Procedure Call Template added the node "Procedure Elaboration Caused Fault" to the template from Leveson, Cha, and Shimeall.**

Function Call Template



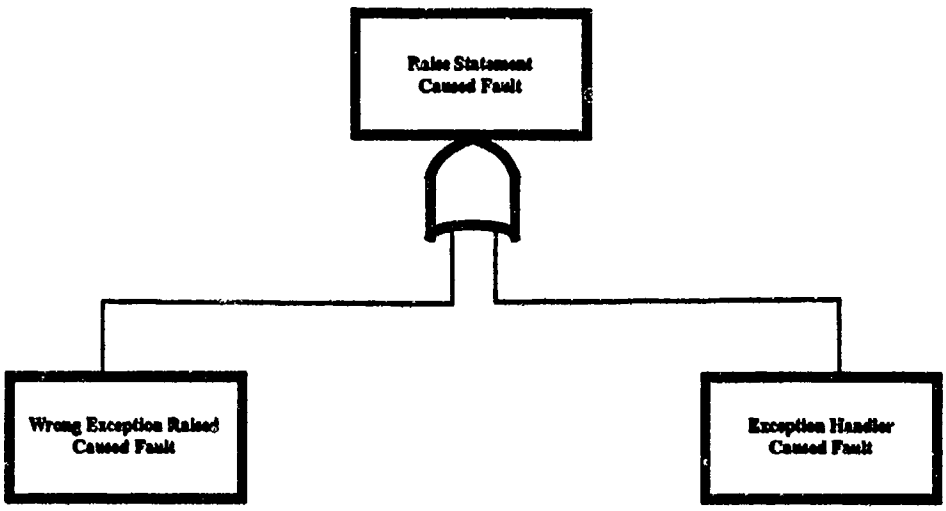
*** The Function Call Template was not depicted in the works of Leveson, Cha, and Shimeall.**

Block Statement Template



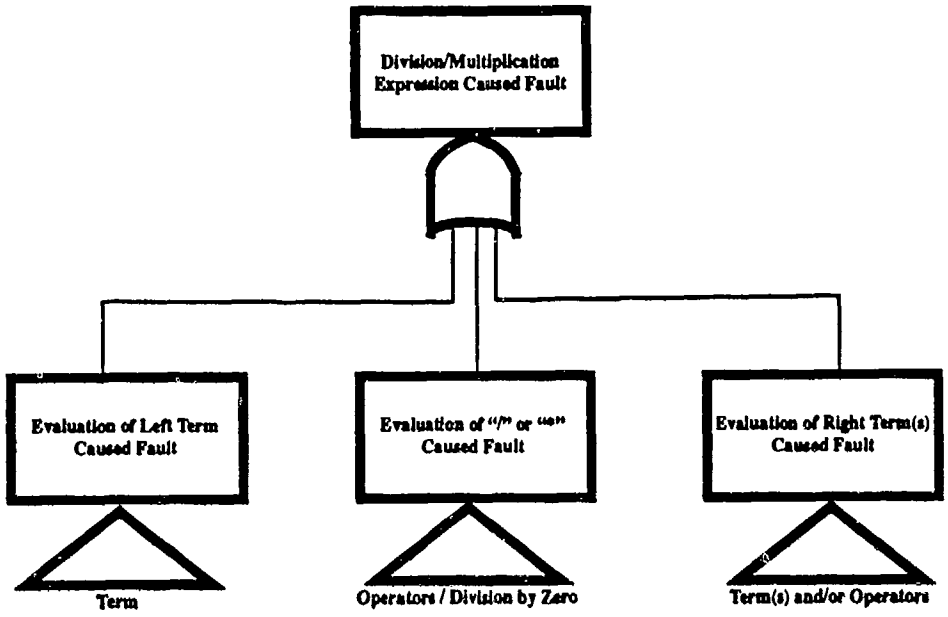
*** The tool generated Block Statement Template elaborated more on the "Exception Caused Fault" node then did the works of Leveson, Cha, and Shimeall.**

Raise Template



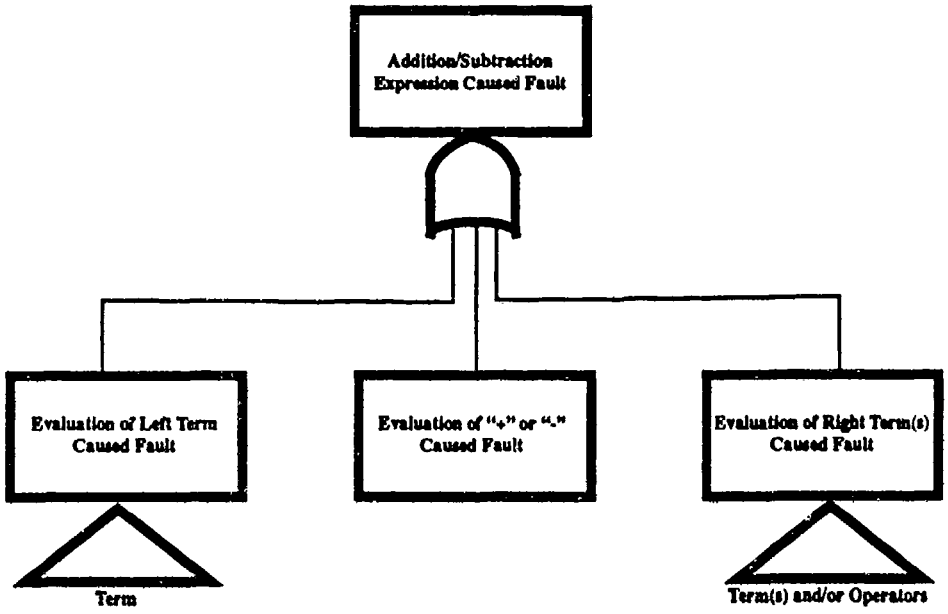
*** There was no difference in the Raise Template between the tool generated and the Leveson, Cha, and Shimeall templates.**

Division and Multiplication Template



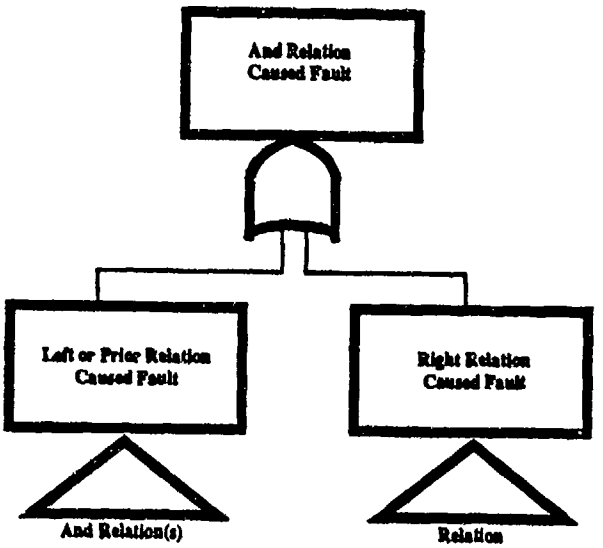
*** Division / Multiplication Template was not depicted in the works of Leveson, Cha, and Shimeall.**

Addition and Subtraction Template



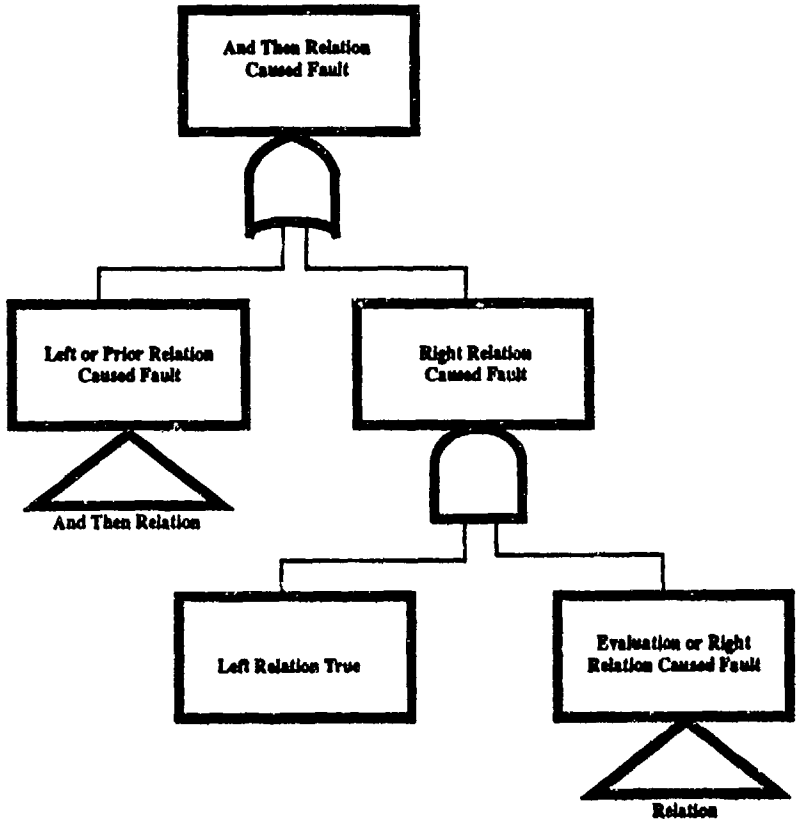
*** Addition / Subtraction Template was not depicted in the works of Leveson, Cha, and Shimeall.**

And Relation Template



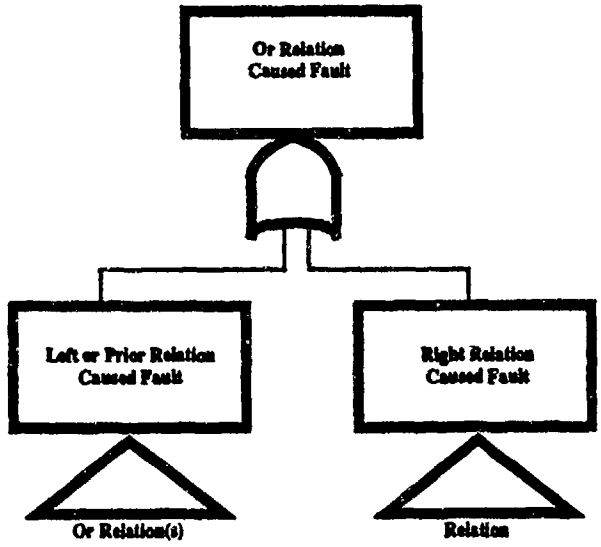
*** And Relation Template was not depicted in the works of Leveson, Cha, and Shimeall.**

And Then Relation Template



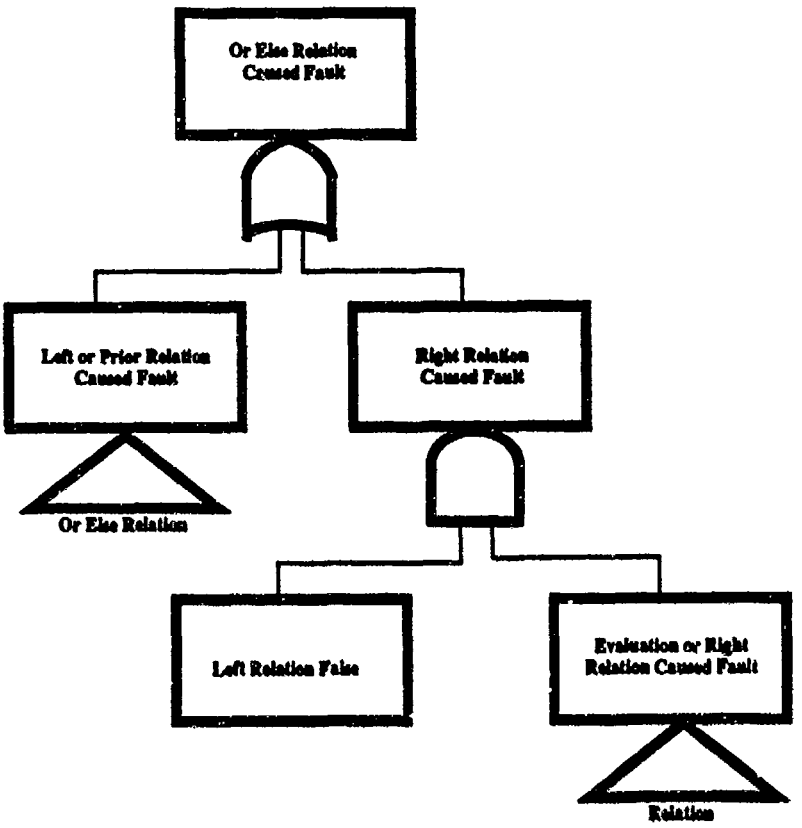
*** And Then Relation Template was not depicted in the works of Leveson, Cha, and Shimeall.**

Or Relation Template



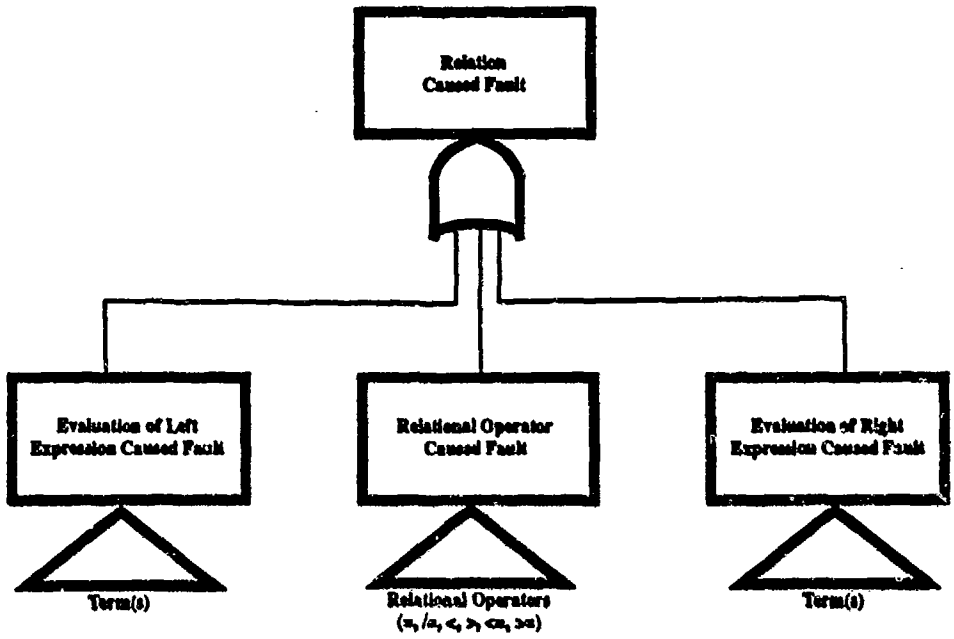
*** Or Relation Template was not depicted in the works of Leveson, Cha, and Shimeall.**

Or Else Relation Template



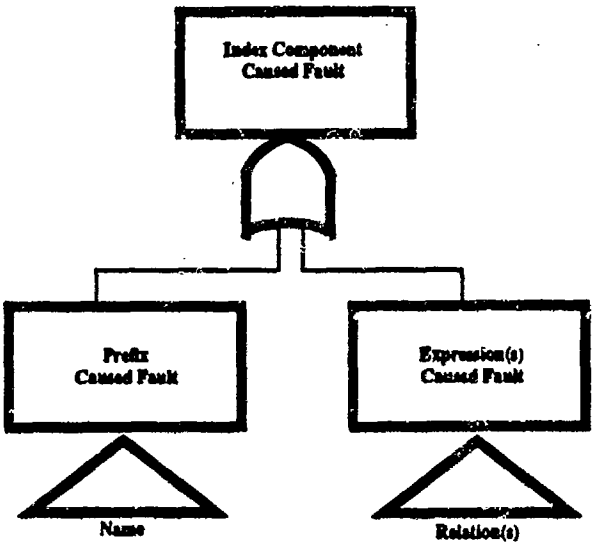
*** Or Else Relation Template was not depicted in the works of Leveson, Cha, and Shimeall.**

Relation Template



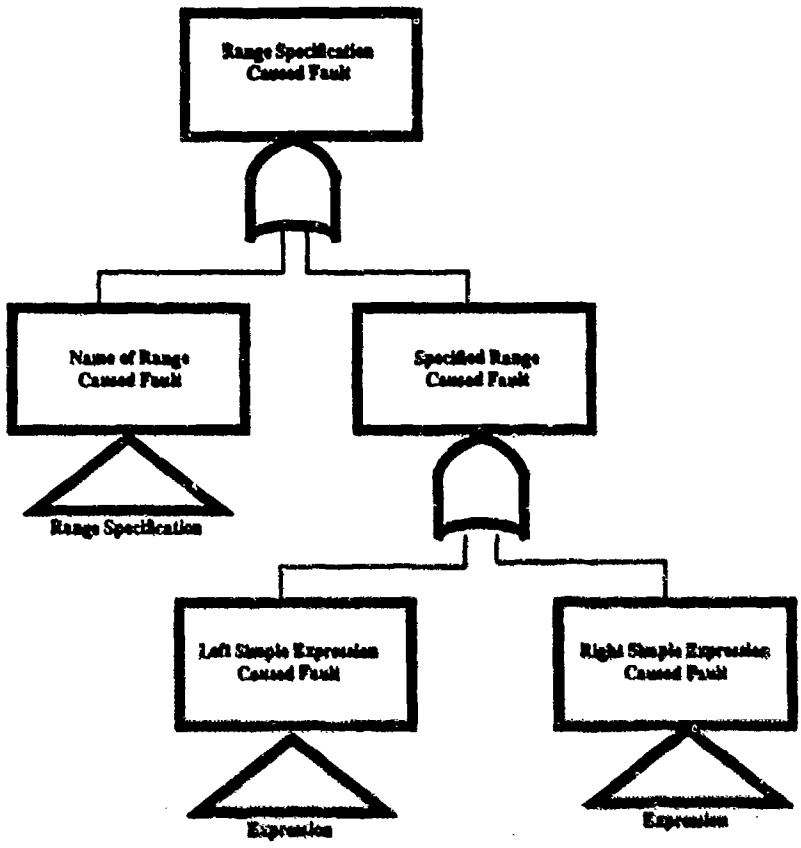
* Relation Template was not depicted in the works of Leveson, Cha, and Shimeall.

Index Component Template



*** Index Component Template was not depicted in the works of Leveson, Cha, and Shimeall.**

Range Template



*** Range Template was not depicted in the works of Leveson, Cha, and Shimeall.**

APPENDIX D: AUTOMATED CODE TRANSLATION TOOL OUTPUT FOR PROCEDURE FAKE_OVEN_CONTROL

A. LITERAL TREE REPRESENTATION

LITERAL TREE OUTPUT

Parent		Parent Node	GATE	Fault for Node
	=>	0	Null	Sequence of statements caused fault
0	=>	1	Or	Last statement caused fault
0	=>	2	Or	Previous statements caused fault
1	=>	3	Null	Loop Statement caused fault
2	=>	4	And	Last Statement did not mask fault
2	=>	5	And	Sequence prior to last caused fault
3	=>	6	Or	Loop never executed
3	=>	7	Or	Loop condition evaluation caused fault
3	=>	8	Or	Nth Iteration caused fault
7	=>	9	Or	Wrong Type of loop used
7	=>	10	Or	Iteration scheme caused fault
8	=>	11	And	Sequence of statements caused fault
8	=>	12	And	Condition true past n-1
10	=>	13	Or	Parameter specification caused fault
11	=>	14	Null	Sequence of statements caused fault
12	=>	15	And	Condition true at n-1 iteration
12	=>	16	And	Sequence of statements kept condition true
13	=>	17	Or	Identifier specified caused fault
13	=>	18	Or	Discreet range caused fault
14	=>	19	Or	Last statement caused fault
14	=>	20	Or	Previous statements caused fault
16	=>	21	Null	Sequence of statements caused fault
17	=>	22	Null	SAMPLE
18	=>	23	Null	Range specified caused fault
19	=>	24	Null	Loop Statement caused fault
20	=>	25	And	Last Statement did not mask fault
20	=>	26	And	Sequence prior to last caused fault
21	=>	27	Or	Last statement caused fault
21	=>	28	Or	Previous statements caused fault
23	=>	29	Or	1
23	=>	30	Or	SAMPLING_DURATION
24	=>	31	Or	Loop never executed
24	=>	32	Or	Loop condition evaluation caused fault
24	=>	33	Or	Nth Iteration caused fault

26 => 34 Null Sequence of statements caused fault
27 => 35 Null Loop Statement caused fault
28 => 36 And Last Statement did not mask fault
28 => 37 And Sequence prior to last caused fault
32 => 38 Or Wrong Type of loop used
32 => 39 Or Iteration scheme caused fault
33 => 40 And Sequence of statements caused fault
33 => 41 And Condition true past n-1
34 => 42 Or Last statement caused fault
34 => 43 Or Previous statements caused fault
35 => 44 Or Loop never executed
35 => 45 Or Loop condition evaluation caused fault
35 => 46 Or Nth Iteration caused fault
37 => 47 Null Sequence of statements caused fault
39 => 48 Or Parameter specification caused fault
41 => 49 And Condition true at n-1 iteration
41 => 50 And Sequence of statements kept condition true
42 => 51 Null If statement caused fault
43 => 52 And Last Statement did not mask fault
43 => 53 And Sequence prior to last caused fault
45 => 54 Or Wrong Type of loop used
45 => 55 Or Iteration scheme caused fault
46 => 56 And Sequence of statements caused fault
46 => 57 And Condition true past n-1
47 => 58 Or Last statement caused fault
47 => 59 Or Previous statements caused fault
48 => 60 Or Identifier specified caused fault
48 => 61 Or Discreet range caused fault
51 => 62 Or Evaluation of condition caused fault
51 => 63 Or Condition true and statements caused fault
51 => 64 Or ELSIF caused the fault
55 => 65 Or Parameter specification caused fault
57 => 66 And Condition true at n-1 iteration
57 => 67 And Sequence of statements kept condition true
58 => 68 Null If statement caused fault
59 => 69 And Last Statement did not mask fault
59 => 70 And Sequence prior to last caused fault
60 => 71 Null I
61 => 72 Null Range specified caused fault
62 => 73 Null Relation caused the fault
63 => 74 And If condition true
63 => 75 And If statements caused fault
64 => 76 And Previous conditions evaluated to false
64 => 77 And Current/future ELSIF caused the fault
65 => 78 Or Identifier specified caused fault
65 => 79 Or Discreet range caused fault
68 => 80 Or Evaluation of condition caused fault
68 => 81 Or Condition true and statements caused fault
68 => 82 Or ELSIF caused the fault
72 => 83 Or 1
72 => 84 Or SAMPLING_DURATION

73 => 85 Null And Relation caused the fault
75 => 86 Null Sequence of statements caused fault
77 => 87 Or Evaluation of Elsif condition caused fault
77 => 88 Or Current ELSIF caused the fault
77 => 89 Or Other ELSIF's caused the fault
78 => 90 Null I
79 => 91 Null Range specified caused fault
80 => 92 Null Relation caused the fault
81 => 93 And If condition true
81 => 94 And If statements caused fault
82 => 95 And Previous conditions evaluated to false
82 => 96 And Current/future ELSIF caused the fault
85 => 97 Or Left or Prior relation caused fault
85 => 98 Or Right relation caused fault
86 => 99 Or Last statement caused fault
86 => 100 Or Previous statements caused fault
87 => 101 Null Relation caused the fault
88 => 102 And Current ELSIF condition caused the fault
88 => 103 And Current ELSIF sequence of statements caused the fault
89 => 104 Null ELSIF caused the fault
91 => 105 Or 1
91 => 106 Or SAMPLING_DURATION
92 => 107 Null And Relation caused the fault
94 => 108 Null Sequence of statements caused fault
96 => 109 Or Evaluation of Elsif condition caused fault
96 => 110 Or Current ELSIF caused the fault
96 => 111 Or Other ELSIF's caused the fault
97 => 112 Null Relation caused the fault
98 => 113 Null Relation caused the fault
99 => 114 Null Assignment Statement Fault
100 => 115 And Last Statement did not mask fault
100 => 116 And Sequence prior to last caused fault
101 => 117 Null Relation caused the fault
103 => 118 Null Sequence of statements caused fault
104 => 119 And Previous conditions evaluated to false
104 => 120 And Current/future ELSIF caused the fault
107 => 121 Or Left or Prior relation caused fault
107 => 122 Or Right relation caused fault
108 => 123 Or Last statement caused fault
108 => 124 Or Previous statements caused fault
109 => 125 Null Relation caused the fault
110 => 126 And Current ELSIF condition caused the fault
110 => 127 And Current ELSIF sequence of statements caused the fault
111 => 128 Null ELSIF caused the fault
112 => 129 Or OVEN_TEMP
112 => 130 Or Addition/Subtraction Fault
113 => 131 Or LAST_CMD
113 => 132 Or 1
114 => 133 Or Change in values caused Fault
114 => 134 Or Exception causes Fault -- Not implemented
114 => 135 Or Operand Evaluation causes Fault

116 => 136 Null Sequence of statements caused fault
117 => 137 Or OVEN_TEMP
117 => 138 Or Addition/Subtraction Fault
118 => 139 Or Last statement caused fault
118 => 140 Or Previous statements caused fault
120 => 141 Or Evaluation of Elsif condition caused fault
120 => 142 Or Current ELSIF caused the fault
120 => 143 Or Other ELSIF's caused the fault
121 => 144 Null Relation caused the fault
122 => 145 Null Relation caused the fault
123 => 146 Null Assignment Statement Fault
124 => 147 And Last Statement did not mask fault
124 => 148 And Sequence prior to last caused fault
125 => 149 Null Relation caused the fault
127 => 150 Null Sequence of statements caused fault
128 => 151 And Previous conditions evaluated to false
128 => 152 And Current/future ELSIF caused the fault
130 => 153 Or GOAL_TEMP
130 => 154 Or Subtraction Fault
130 => 155 Or 10
135 => 156 Or LAST_CMD
135 => 157 Or Relation caused the fault
136 => 158 Or Last statement caused fault
136 => 159 Or Previous statements caused fault
138 => 160 Or GOAL_TEMP
138 => 161 Or Addition Fault
138 => 162 Or 10
139 => 163 Null Raise Statement Caused Fault
140 => 164 And Last Statement did not mask fault
140 => 165 And Sequence prior to last caused fault
141 => 166 Null Relation caused the fault
142 => 167 And Current ELSIF condition caused the fault
142 => 168 And Current ELSIF sequence of statements caused the fault
143 => 169 Null ELSIF caused the fault
144 => 170 Or OVEN_TEMP
144 => 171 Or Addition/Subtraction Fault
145 => 172 Or LAST_CMD
145 => 173 Or 1
146 => 174 Or Change in values caused Fault
146 => 175 Or Exception causes Fault -- Not implemented
146 => 176 Or Operand Evaluation causes Fault
148 => 177 Null Sequence of statements caused fault
149 => 178 Or OVEN_TEMP
149 => 179 Or Addition/Subtraction Fault
150 => 180 Or Last statement caused fault
150 => 181 Or Previous statements caused fault
152 => 182 Or Evaluation of Elsif condition caused fault
152 => 183 Or Current ELSIF caused the fault
152 => 184 Or Other ELSIF's caused the fault
157 => 185 Null 1
158 => 186 Null Assignment Statement Fault

159 => 187 And Last Statement did not mask fault
159 => 188 And Sequence prior to last caused fault
163 => 189 Or Wrong Exception Raised Caused Fault
163 => 190 Or Exception Handler Caused Fault
166 => 191 Null And Relation caused the fault
168 => 192 Null Sequence of statements caused fault
169 => 193 And Previous conditions evaluated to false
169 => 194 And Current/future ELSIF caused the fault
171 => 195 Or GOAL_TEMP
171 => 196 Or Subtraction Fault
171 => 197 Or 10
176 => 198 Or LAST_CMD
176 => 199 Or Relation caused the fault
177 => 200 Or Last statement caused fault
177 => 201 Or Previous statements caused fault
179 => 202 Or GOAL_TEMP
179 => 203 Or Addition Fault
179 => 204 Or 10
180 => 205 Null Raise Statement Caused Fault
181 => 206 And Last Statement did not mask fault
181 => 207 And Sequence prior to last caused fault
182 => 208 Null Relation caused the fault
183 => 209 And Current ELSIF condition caused the fault
183 => 210 And Current ELSIF sequence of statements caused the fault
184 => 211 Null ELSIF caused the fault
186 => 212 Or Change in values caused Fault
186 => 213 Or Exception causes Fault -- Not implemented
186 => 214 Or Operand Evaluation causes Fault
190 => 215 Null OVEN_NOT_RESPONDING
191 => 216 Or Left or Prior relation caused fault
191 => 217 Or Right relation caused fault
192 => 218 Or Last statement caused fault
192 => 219 Or Previous statements caused fault
194 => 220 Or Evaluation of Elsif condition caused fault
194 => 221 Or Current ELSIF caused the fault
194 => 222 Or Other ELSIF's caused the fault
199 => 223 Null 1
200 => 224 Null Assignment Statement Fault
201 => 225 And Last Statement did not mask fault
201 => 226 And Sequence prior to last caused fault
205 => 227 Or Wrong Exception Raised Caused Fault
205 => 228 Or Exception Handler Caused Fault
208 => 229 Null And Relation caused the fault
210 => 230 Null Sequence of statements caused fault
211 => 231 And Previous conditions evaluated to false
211 => 232 And Current/future ELSIF caused the fault
214 => 233 Or OVEN_BUF
214 => 234 Or Relation caused the fault
216 => 235 Null Relation caused the fault
217 => 236 Null Relation caused the fault
218 => 237 Null Assignment Statement Fault

219 => 238 And Last Statement did not mask fault
219 => 239 And Sequence prior to last caused fault
220 => 240 Null Relation caused the fault
221 => 241 And Current ELSIF condition caused the fault
221 => 242 And Current ELSIF sequence of statements caused the fault
224 => 243 Or Change in values caused Fault
224 => 244 Or Exception causes Fault -- Not implemented
224 => 245 Or Operand Evaluation causes Fault
228 => 246 Null OVEN_NOT_RESPONDING
229 => 247 Or Left or Prior relation caused fault
229 => 248 Or Right relation caused fault
230 => 249 Or Last statement caused fault
230 => 250 Or Previous statements caused fault
232 => 251 Or Evaluation of Elsif condition caused fault
232 => 252 Or Current ELSIF caused the fault
232 => 253 Or Other ELSIF's caused the fault
234 => 254 Null 1
235 => 255 Or OVEN_TEMP
235 => 256 Or Addition/Subtraction Fault
236 => 257 Or LAST_CMD
236 => 258 Or 2
237 => 259 Or Change in values caused Fault
237 => 260 Or Exception causes Fault -- Not implemented
237 => 261 Or Operand Evaluation causes Fault
239 => 262 Null Sequence of statements caused fault
240 => 263 Null Relation caused the fault
242 => 264 Null Sequence of statements caused fault
245 => 265 Or OVEN_BUF
245 => 266 Or Relation caused the fault
247 => 267 Null Relation caused the fault
248 => 268 Null Relation caused the fault
249 => 269 Null Assignment Statement Fault
250 => 270 And Last Statement did not mask fault
250 => 271 And Sequence prior to last caused fault
251 => 272 Null Relation caused the fault
252 => 273 And Current ELSIF condition caused the fault
252 => 274 And Current ELSIF sequence of statements caused the fault
253 => 275 Or GOAL_TEMP
256 => 276 Or Addition Fault
256 => 277 Or 10
261 => 278 Or LAST_CMD
261 => 279 Or Relation caused the fault
262 => 280 Or Last statement caused fault
262 => 281 Or Previous statements caused fault
263 => 282 Or OVEN_TEMP
263 => 283 Or Addition/Subtraction Fault
264 => 284 Or Last statement caused fault
264 => 285 Or Previous statements caused fault
266 => 286 Null 1
267 => 287 Or OVEN_TEMP
267 => 288 Or Addition/Subtraction Fault

268 => 289 Or LAST_CMD
268 => 290 Or 2
269 => 291 Or Change in values caused Fault
269 => 292 Or Exception causes Fault -- Not implemented
269 => 293 Or Operand Evaluation causes Fault
271 => 294 Null Sequence of statements caused fault
272 => 295 Null Relation caused the fault
274 => 296 Null Sequence of statements caused fault
279 => 297 Null 2
280 => 298 Null Assignment Statement Fault
281 => 299 And Last Statement did not mask fault
281 => 300 And Sequence prior to last caused fault
283 => 301 Or GOAL_TEMP
283 => 302 Or Subtraction Fault
283 => 303 Or 10
284 => 304 Null Raise Statement Caused Fault
285 => 305 And Last Statement did not mask fault
285 => 306 And Sequence prior to last caused fault
288 => 307 Or GOAL_TEMP
288 => 308 Or Addition Fault
288 => 309 Or 10
293 => 310 Or LAST_CMD
293 => 311 Or Relation caused the fault
294 => 312 Or Last statement caused fault
294 => 313 Or Previous statements caused fault
295 => 314 Or OVEN_TEMP
295 => 315 Or Addition/Subtraction Fault
296 => 316 Or Last statement caused fault
296 => 317 Or Previous statements caused fault
298 => 318 Or Change in values caused Fault
298 => 319 Or Exception causes Fault -- Not implemented
298 => 320 Or Operand Evaluation causes Fault
304 => 321 Or Wrong Exception Raised Caused Fault
304 => 322 Or Exception Handler Caused Fault
311 => 323 Null 2
312 => 324 Null Assignment Statement Fault
313 => 325 And Last Statement did not mask fault
313 => 326 And Sequence prior to last caused fault
315 => 327 Or GOAL_TEMP
315 => 328 Or Subtraction Fault
315 => 329 Or 10
316 => 330 Null Raise Statement Caused Fault
317 => 331 And Last Statement did not mask fault
317 => 332 And Sequence prior to last caused fault
320 => 333 Or OVEN_BUF
320 => 334 Or Relation caused the fault
322 => 335 Null OVEN_NOT_RESPONDING
324 => 336 Or Change in values caused Fault
324 => 337 Or Exception causes Fault -- Not implemented
324 => 338 Or Operand Evaluation causes Fault
330 => 339 Or Wrong Exception Raised Caused Fault

330 => 340 Or Exception Handler Caused Fault
334 => 341 Null 2
338 => 342 Or OVEN_BUF
338 => 343 Or Relation caused the fault
340 => 344 Null OVEN_NOT_RESPONDING
343 => 345 Null 2

B. FTE NODE LISTING

FTE FILE OUTPUT

Node	Fault for Node
0	Sequence of statements caused fault
1	Last statement caused fault
2	Previous statements caused fault
3	Loop Statement caused fault
4	Last Statement did not mask fault
5	Sequence prior to last caused fault
6	Loop never executed
7	Loop condition evaluation caused fault
8	Nth Iteration caused fault
9	Wrong Type of loop used
10	Iteration scheme caused fault
11	Sequence of statements caused fault
12	Condition true past n-1
13	Parameter specification caused fault
14	Sequence of statements caused fault
15	Condition true at n-1 iteration
16	Sequence of statements kept condition true
17	Identifier specified caused fault
18	Discreet range caused fault
19	Last statement caused fault
20	Previous statements caused fault
21	Sequence of statements caused fault
22	SAMPLE
23	Range specified caused fault
24	Loop Statement caused fault
25	Last Statement did not mask fault
26	Sequence prior to last caused fault
27	Last statement caused fault
28	Previous statements caused fault
29	1
30	SAMPLING_DURATION
31	Loop never executed
32	Loop condition evaluation caused fault
33	Nth Iteration caused fault
34	Sequence of statements caused fault
35	Loop Statement caused fault
36	Last Statement did not mask fault
37	Sequence prior to last caused fault
38	Wrong Type of loop used
39	Iteration scheme caused fault
40	Sequence of statements caused fault
41	Condition true past n-1

42 Last statement caused fault
43 Previous statements caused fault
44 Loop never executed
45 Loop condition evaluation caused fault
46 Nth Iteration caused fault
47 Sequence of statements caused fault
48 Parameter specification caused fault
49 Condition true at n-1 iteration
50 Sequence of statements kept condition true
51 If statement caused fault
52 Last Statement did not mask fault
53 Sequence prior to last caused fault
54 Wrong Type of loop used
55 Iteration scheme caused fault
56 Sequence of statements caused fault
57 Condition true past n-1
58 Last statement caused fault
59 Previous statements caused fault
60 Identifier specified caused fault
61 Discreet range caused fault
62 Evaluation of condition caused fault
63 Condition true and statements caused fault
64 ELSIF caused the fault
65 Parameter specification caused fault
66 Condition true at n-1 iteration
67 Sequence of statements kept condition true
68 If statement caused fault
69 Last Statement did not mask fault
70 Sequence prior to last caused fault
71 I
72 Range specified caused fault
73 Relation caused the fault
74 If condition true
75 If statements caused fault
76 Previous conditions evaluated to false
77 Current/future ELSIF caused the fault
78 Identifier specified caused fault
79 Discreet range caused fault
80 Evaluation of condition caused fault
81 Condition true and statements caused fault
82 ELSIF caused the fault
83 1
84 SAMPLING_DURATION
85 And Relation caused the fault
86 Sequence of statements caused fault
87 Evaluation of Elsif condition caused fault
88 Current ELSIF caused the fault
89 Other ELSIF's caused the fault
90 1
91 Range specified caused fault
92 Relation caused the fault

93 If condition true
94 If statements caused fault
95 Previous conditions evaluated to false
96 Current/future ELSIF caused the fault
97 Left or Prior relation caused fault
98 Right relation caused fault
99 Last statement caused fault
100 Previous statements caused fault
101 Relation caused the fault
102 Current ELSIF condition caused the fault
103 Current ELSIF sequence of statements caused the fault
104 ELSIF caused the fault
105 1
106 SAMPLING_DURATION
107 And Relation caused the fault
108 Sequence of statements caused fault
109 Evaluation of Elsif condition caused fault
110 Current ELSIF caused the fault
111 Other ELSIF's caused the fault
112 Relation caused the fault
113 Relation caused the fault
114 Assignment Statement Fault
115 Last Statement did not mask fault
116 Sequence prior to last caused fault
117 Relation caused the fault
118 Sequence of statements caused fault
119 Previous conditions evaluated to false
120 Current/future ELSIF caused the fault
121 Left or Prior relation caused fault
122 Right relation caused fault
123 Last statement caused fault
124 Previous statements caused fault
125 Relation caused the fault
126 Current ELSIF condition caused the fault
127 Current ELSIF sequence of statements caused the fault
128 ELSIF caused the fault
129 OVEN_TEMP
130 Addition/Subtraction Fault
131 ^ST_CMD
132 1
133 Change in values caused Fault
134 Exception causes Fault -- Not implemented
135 Operand Evaluation causes Fault
136 Sequence of statements caused fault
137 OVEN_TEMP
138 Addition/Subtraction Fault
139 Last statement caused fault
140 Previous statements caused fault
141 Evaluation of Elsif condition caused fault
142 Current ELSIF caused the fault
143 Other ELSIF's caused the fault

144 Relation caused the fault
145 Relation caused the fault
146 Assignment Statement Fault
147 Last Statement did not mask fault
148 Sequence prior to last caused fault
149 Relation caused the fault
150 Sequence of statements caused fault
151 Previous conditions evaluated to false
152 Current/future ELSIF caused the fault
153 GOAL_TEMP
154 Subtraction Fault
155 10
156 LAST_CMD
157 Relation caused the fault
158 Last statement caused fault
159 Previous statements caused fault
160 GOAL_TEMP
161 Addition Fault
162 10
163 Raise Statement Caused Fault
164 Last Statement did not mask fault
165 Sequence prior to last caused fault
166 Relation caused the fault
167 Current ELSIF condition caused the fault
168 Current ELSIF sequence of statements caused the fault
169 ELSIF caused the fault
170 OVEN_TEMP
171 Addition/Subtraction Fault
172 LAST_CMD
173 1
174 Change in values caused Fault
175 Exception causes Fault -- Not implemented
176 Operand Evaluation causes Fault
177 Sequence of statements caused fault
178 OVEN_TEMP
179 Addition/Subtraction Fault
180 Last statement caused fault
181 Previous statements caused fault
182 Evaluation of Elsif condition caused fault
183 Current ELSIF caused the fault
184 Other ELSIF's caused the fault
185 1
186 Assignment Statement Fault
187 Last Statement did not mask fault
188 Sequence prior to last caused fault
189 Wrong Exception Raised Caused Fault
190 Exception Handler Caused Fault
191 And Relation caused the fault
192 Sequence of statements caused fault
193 Previous conditions evaluated to false
194 Current/future ELSIF caused the fault

195 GOAL_TEMP
196 Subtraction Fault
197 10
198 LAST_CMD
199 Relation caused the fault
200 Last statement caused fault
201 Previous statements caused fault
202 GOAL_TEMP
203 Addition Fault
204 10
205 Raise Statement Caused Fault
206 Last Statement did not mask fault
207 Sequence prior to last caused fault
208 Relation caused the fault
209 Current ELSIF condition caused the fault
210 Current ELSIF sequence of statements caused the fault
211 ELSIF caused the fault
212 Change in values caused Fault
213 Exception causes Fault -- Not implemented
214 Operand Evaluation causes Fault
215 OVEN_NOT_RESPONDING
216 Left or Prior relation caused fault
217 Right relation caused fault
218 Last statement caused fault
219 Previous statements caused fault
220 Evaluation of Elsif condition caused fault
221 Current ELSIF caused the fault
222 Other ELSIF's caused the fault
223 1
224 Assignment Statement Fault
225 Last Statement did not mask fault
226 Sequence prior to last caused fault
227 Wrong Exception Raised Caused Fault
228 Exception Handler Caused Fault
229 And Relation caused the fault
230 Sequence of statements caused fault
231 Previous conditions evaluated to false
232 Current/future ELSIF caused the fault
233 OVEN_BUF
234 Relation caused the fault
235 Relation caused the fault
236 Relation caused the fault
237 Assignment Statement Fault
238 Last Statement did not mask fault
239 Sequence prior to last caused fault
240 Relation caused the fault
241 Current ELSIF condition caused the fault
242 Current ELSIF sequence of statements caused the fault
243 Change in values caused Fault
244 Exception causes Fault -- Not implemented
245 Operand Evaluation causes Fault

246 OVEN_NOT_RESPONDING
247 Left or Prior relation caused fault
248 Right relation caused fault
249 Last statement caused fault
250 Previous statements caused fault
251 Evaluation of Elsif condition caused fault
252 Current ELSIF caused the fault
253 Other ELSIF's caused the fault
254 1
255 OVEN_TEMP
256 Addition/Subtraction Fault
257 LAST_CMD
258 2
259 Change in values caused Fault
260 Exception causes Fault -- Not implemented
261 Operand Evaluation causes Fault
262 Sequence of statements caused fault
263 Relation caused the fault
264 Sequence of statements caused fault
265 OVEN_BUF
266 Relation caused the fault
267 Relation caused the fault
268 Relation caused the fault
269 Assignment Statement Fault
270 Last Statement did not mask fault
271 Sequence prior to last caused fault
272 Relation caused the fault
273 Current ELSIF condition caused the fault
274 Current ELSIF sequance of statements caused the fault
275 GOAL_TEMP
276 Addition Fault
277 10
278 LAST_CMD
279 Relation caused the fault
280 Last statement caused fault
281 Previous statements caused fault
282 OVEN_TEMP
283 Addition/Subtraction Fault
284 Last statement caused fault
285 Previous statements caused fault
286 1
287 OVEN_TEMP
288 Addition/Subtraction Fault
289 LAST_CMD
290 2
291 Change in values caused Fault
292 Exception causes Fault -- Not implemented
293 Operand Evaluation causes Fault
294 Sequence of statements caused fault
295 Relation caused the fault
296 Sequence of statements caused fault

297 2
298 Assignment Statement Fault
299 Last Statement did not mask fault
300 Sequence prior to last caused fault
301 GOAL_TEMP
302 Subtraction Fault
303 10
304 Raise Statement Caused Fault
305 Last Statement did not mask fault
306 Sequence prior to last caused fault
307 GOAL_TEMP
308 Addition Fault
309 10
310 LAST_CMD
311 Relation caused the fault
312 Last statement caused fault
313 Previous statements caused fault
314 OVEN_TEMP
315 Addition/Subtraction Fault
316 Last statement caused fault
317 Previous statements caused fault
318 Change in values caused Fault
319 Exception causes Fault -- Not implemented
320 Operand Evaluation causes Fault
321 Wrong Exception Raised Caused Fault
322 Exception Handler Caused Fault
323 2
324 Assignment Statement Fault
325 Last Statement did not mask fault
326 Sequence prior to last caused fault
327 GOAL_TEMP
328 Subtraction Fault
329 10
330 Raise Statement Caused Fault
331 Last Statement did not mask fault
332 Sequence prior to last caused fault
333 OVEN_BUF
334 Relation caused the fault
335 OVEN_NOT_RESPONDING
336 Change in values caused Fault
337 Exception causes Fault -- Not implemented
338 Operand Evaluation causes Fault
339 Wrong Exception Raised Caused Fault
340 Exception Handler Caused Fault
341 2
342 OVEN_BUF
343 Relation caused the fault
344 OVEN_NOT_RESPONDING
345 2

C. PROCEDURE, FUNCTION, AND EXCEPTION LISTING

The number of procedures/functions on the table are 1

The procedures/functions and their root faults on the table
are the following:

FAKE_OVEN_CONTROL
Sequence of statements caused fault

Above are the procedures/functions and root faults.

++

There are no defined exceptions within the input code.

---- Finished Code Translation ----

0 errors found

D. FTE COMPATIBLE FILE, "NEW_FTE"

```
 0
Sequence of statements caused fault
EXAMPLE.A
0 0 0 0 1 2 2
 1
Last statement caused fault
EXAMPLE.A
0 0 0 85 1 0 1
 3
Loop Statement caused fault
EXAMPLE.A
0 0 0 170 1 2 3
 6
Loop never executed
EXAMPLE.A
0 0 0 255 1 0 0
 7
Loop condition evaluation caused fault
EXAMPLE.A
0 0 75 255 1 2 2
 9
Wrong Type of loop used
EXAMPLE.A
0 0 0 340 1 0 0
 10
Iteration scheme caused fault
EXAMPLE.A
0 0 75 340 1 2 1
 13
Parameter specification caused fault
EXAMPLE.A
0 0 0 425 1 2 2
 17
Identifier specified caused fault
EXAMPLE.A
0 0 0 510 1 0 1
 22
SAMPLE
EXAMPLE.A
0 0 0 595 1 0 0
 18
Discreet range caused fault
EXAMPLE.A
0 0 75 510 1 0 1
 23
Range specified caused fault
EXAMPLE.A
0 0 75 595 1 2 2
```

29
1
EXAMPLE.A
0 0 0 680 1 0 0
30
SAMPLING_DURATION
EXAMPLE.A
0 0 75 680 1 0 0
8
Nth Iteration caused fault
EXAMPLE.A
0 0 150 255 1 1 2
11
Sequence of statements caused fault
EXAMPLE.A
0 0 150 340 1 0 1
21
Sequence of statements caused fault
EXAMPLE.A
0 0 300 510 1 2 2
27
Last statement caused fault
EXAMPLE.A
0 0 375 595 1 0 1
35
Loop Statement caused fault
EXAMPLE.A
0 0 450 680 1 2 3
44
Loop never executed
EXAMPLE.A
0 0 450 765 1 0 0
45
Loop condition evaluation caused fault
EXAMPLE.A
0 0 525 765 1 2 2
54
Wrong Type of loop used
EXAMPLE.A
0 0 450 850 1 0 0
55
Iteration scheme caused fault
EXAMPLE.A
0 0 525 850 1 2 1
65
Parameter specification caused fault
EXAMPLE.A
0 0 375 935 1 2 2
78
Identifier specified caused fault
EXAMPLE.A

0 0 525 1020 1 0 1
90
I
EXAMPLE.A
0 0 525 1105 1 0 0
79
Discreet range caused fault
EXAMPLE.A
0 0 600 1020 1 0 1
91
Range specified caused fault
EXAMPLE.A
0 0 600 1105 1 2 2
105
1
EXAMPLE.A
0 0 600 1190 1 0 0
106
SAMPLING_DURATION
EXAMPLE.A
0 0 675 1190 1 0 0
46
Nth Iteration caused fault
EXAMPLE.A
0 0 600 765 1 1 2
56
Sequence of statements caused fault
EXAMPLE.A
0 0 600 850 1 0 1
57
Condition true past n-1
EXAMPLE.A
0 0 675 850 1 1 2
66
Condition true at n-1 iteration
EXAMPLE.A
0 0 450 935 1 0 0
67
Sequence of statements kept condition true
EXAMPLE.A
0 0 525 935 1 0 1
28
Previous statements caused fault
EXAMPLE.A
0 0 450 595 1 1 2
36
Last Statement did not mask fault
EXAMPLE.A
0 0 525 680 1 0 0
37
Sequence prior to last caused fault

EXAMPLE.A
0 0 600 680 1 0 1
47
Sequence of statements caused fault
EXAMPLE.A
0 0 675 765 1 2 2
58
Last statement caused fault
EXAMPLE.A
0 0 750 850 1 0 1
68
If statement caused fault
EXAMPLE.A
0 0 600 935 1 2 3
80
Evaluation of condition caused fault
EXAMPLE.A
0 0 675 1020 1 0 0
92
Relation caused the fault
EXAMPLE.A
0 0 675 1105 1 0 1
107
And Relation caused the fault
EXAMPLE.A
0 0 750 1190 1 2 2
121
Left or Prior relation caused fault
EXAMPLE.A
0 0 675 1275 1 0 1
144
Relation caused the fault
EXAMPLE.A
0 0 1125 1360 1 2 2
170
OVEN_TEMP
EXAMPLE.A
0 0 1275 1445 1 0 0
171
Addition/Subtraction Fault
EXAMPLE.A
0 0 1350 1445 1 2 3
195
GOAL_TEMP
EXAMPLE.A
0 0 750 1530 1 0 0
196
Subtraction Fault
EXAMPLE.A
0 0 825 1530 1 0 0
197

10
EXAMPLE.A
0 0 900 1530 1 0 0
122
Right relation caused fault
EXAMPLE.A
0 0 750 1275 1 0 1
145
Relation caused the fault
EXAMPLE.A
0 0 1200 1360 1 2 2
172
LAST_CMD
EXAMPLE.A
0 0 1425 1445 1 0 0
173
1
EXAMPLE.A
0 0 1500 1445 1 0 0
81
Condition true and statements caused fault
EXAMPLE.A
0 0 750 1020 1 1 2
93
If condition true
EXAMPLE.A
0 0 750 1105 1 0 0
94
If statements caused fault
EXAMPLE.A
0 0 825 1105 1 0 0
108
Sequence of statements caused fault
EXAMPLE.A
0 0 825 1190 1 2 2
123
Last statement caused fault
EXAMPLE.A
0 0 825 1275 1 0 1
146
Assignment Statement Fault
EXAMPLE.A
0 0 1275 1360 1 2 3
174
Change in values caused Fault
EXAMPLE.A
0 0 1575 1445 1 0 0
175
Exception causes Fault -- Not implemented
EXAMPLE.A
0 0 1650 1445 1 0 0

176
Operand Evaluation causes Fault
EXAMPLE.A
0 0 1725 1445 1 2 2
198
LAST_CMD
EXAMPLE.A
0 0 975 1530 1 0 0
199
Relation caused the fault
EXAMPLE.A
0 0 1050 1530 1 0 1
223
1
EXAMPLE.A
0 0 825 1615 1 0 0
124
Previous statements caused fault
EXAMPLE.A
0 0 900 1275 1 1 2
147
Last Statement did not mask fault
EXAMPLE.A
0 0 1350 1360 1 0 0
148
Sequence prior to last caused fault
EXAMPLE.A
0 0 1425 1360 1 0 1
177
Sequence of statements caused fault
EXAMPLE.A
0 0 1800 1445 1 2 2
200
Last statement caused fault
EXAMPLE.A
0 0 1125 1530 1 0 1
224
Assignment Statement Fault
EXAMPLE.A
0 0 900 1615 1 2 3
243
Change in values caused Fault
EXAMPLE.A
0 0 750 1700 1 0 0
244
Exception causes Fault -- Not implemented
EXAMPLE.A
0 0 825 1700 1 0 0
245
Operand Evaluation causes Fault
EXAMPLE.A

0 0 900 1700 1 2 2
265
OVEN_BUF
EXAMPLE.A
0 0 825 1785 1 0 0
266
Relation caused the fault
EXAMPLE.A
0 0 900 1785 1 0 1
286
1
EXAMPLE.A
0 0 825 1870 1 0 0
201
Previous statements caused fault
EXAMPLE.A
0 0 1200 1530 1 1 2
225
Last Statement did not mask fault
EXAMPLE.A
0 0 975 1615 1 0 0
226
Sequence prior to last caused fault
EXAMPLE.A
0 0 1050 1615 1 0 0
82
ELSIF caused the fault
EXAMPLE.A
0 0 825 1020 1 1 2
95
Previous conditions evaluated to false
EXAMPLE.A
0 0 900 1105 1 0 0
96
Current/future ELSIF caused the fault
EXAMPLE.A
0 0 975 1105 1 2 3
109
Evaluation of Elsif condition caused fault
EXAMPLE.A
0 0 900 1190 1 0 0
125
Relation caused the fault
EXAMPLE.A
0 0 975 1275 1 0 1
149
Relation caused the fault
EXAMPLE.A
0 0 1500 1360 1 2 2
178
OVEN_TEMP

EXAMPLE.A
0 0 1875 1445 1 0 0
179
Addition/Subtraction Fault
EXAMPLE.A
0 0 1950 1445 1 2 3
202
GOAL_TEMP
EXAMPLE.A
0 0 1275 1530 1 0 0
203
Addition Fault
EXAMPLE.A
0 0 1350 1530 1 0 0
204
10
EXAMPLE.A
0 0 1425 1530 1 0 0
110
Current ELSIF caused the fault
EXAMPLE.A
0 0 975 1190 1 1 2
126
Current ELSIF condition caused the fault
EXAMPLE.A
0 0 1050 1275 1 0 0
127
Current ELSIF sequence of statements caused the fault
EXAMPLE.A
0 0 1125 1275 1 0 0
150
Sequence of statements caused fault
EXAMPLE.A
0 0 1575 1360 1 2 2
180
Last statement caused fault
EXAMPLE.A
0 0 2025 1445 1 0 1
205
Raise Statement Caused Fault
EXAMPLE.A
0 0 1500 1530 1 2 2
227
Wrong Exception Raised Caused Fault
EXAMPLE.A
0 0 1125 1615 1 0 0
228
Exception Handler Caused Fault
EXAMPLE.A
0 0 1200 1615 1 0 1
246

OVEN_NOT_RESPONDING
EXAMPLE.A
0 0 975 1700 1 0 0
181
Previous statements caused fault
EXAMPLE.A
0 0 2100 1445 1 1 2
206
Last Statement did not mask fault
EXAMPLE.A
0 0 1575 1530 1 0 0
207
Sequence prior to last caused fault
EXAMPLE.A
0 0 1650 1530 1 0 0
111
Other ELSIF's caused the fault
EXAMPLE.A
0 0 1050 1190 1 0 0
128
ELSIF caused the fault
EXAMPLE.A
0 0 1200 1275 1 1 2
151
Previous conditions evaluated to false
EXAMPLE.A
0 0 1650 1360 1 0 0
152
Current/future ELSIF caused the fault
EXAMPLE.A
0 0 1725 1360 1 2 3
182
Evaluation of Elsif condition caused fault
EXAMPLE.A
0 0 2175 1445 1 0 0
208
Relation caused the fault
EXAMPLE.A
0 0 1725 1530 1 0 1
229
And Relation caused the fault
EXAMPLE.A
0 0 1275 1615 1 2 2
247
Left or Prior relation caused fault
EXAMPLE.A
0 0 1050 1700 1 0 1
267
Relation caused the fault
EXAMPLE.A
0 0 975 1785 1 2 2

287
OVEN_TEMP
EXAMPLE.A
0 0 900 1870 1 0 0
288
Addition/Subtraction Fault
EXAMPLE.A
0 0 975 1870 1 2 3
307
GOAL_TEMP
EXAMPLE.A
0 0 750 1955 1 0 0
308
Addition Fault
EXAMPLE.A
0 0 825 1955 1 0 0
309
10
EXAMPLE.A
0 0 900 1955 1 0 0
248
Right relation caused fault
EXAMPLE.A
0 0 1125 1700 1 0 1
268
Relation caused the fault
EXAMPLE.A
0 0 1050 1785 1 2 2
289
LAST_CMD
EXAMPLE.A
0 0 1050 1870 1 0 0
290
2
EXAMPLE.A
0 0 1125 1870 1 0 0
183
Current ELSIF caused the fault
EXAMPLE.A
0 0 2250 1445 1 1 2
209
Current ELSIF condition caused the fault
EXAMPLE.A
0 0 1800 1530 1 0 0
210
Current ELSIF sequence of statements caused the fault
EXAMPLE.A
0 0 1875 1530 1 0 0
230
Sequence of statements caused fault
EXAMPLE.A

0 0 1350 1615 1 2 2
249
Last statement caused fault
EXAMPLE.A
0 0 1200 1700 1 0 1
269
Assignment Statement Fault
EXAMPLE.A
0 0 1125 1785 1 2 3
291
Change in values caused Fault
EXAMPLE.A
0 0 1200 1870 1 0 0
292
Excepion causes Fault – Not implemented
EXAMPLE.A
0 0 1275 1870 1 0 0
293
Operand Evaluation causes Fault
EXAMPLE.A
0 0 1350 1870 1 2 2
310
LAST_CMD
EXAMPLE.A
0 0 975 1955 1 0 0
311
Relation caused the fault
EXAMPLE.A
0 0 1050 1955 1 0 1
323
2
EXAMPLE.A
0 0 375 2040 1 0 0
250
Previous statements caused fault
EXAMPLE.A
0 0 1275 1700 1 1 2
270
Last Statement did not mask fault
EXAMPLE.A
0 0 1200 1785 1 0 0
271
Sequence prior to last caused fault
EXAMPLE.A
0 0 1275 1785 1 0 1
294
Sequence of statements caused fault
EXAMPLE.A
0 0 1425 1870 1 2 2
312
Last statement caused fault

EXAMPLE.A
0 0 1125 1955 1 0 1
324
Assignment Statement Fault
EXAMPLE.A
0 0 450 2040 1 2 3
336
Change in values caused Fault
EXAMPLE.A
0 0 225 2125 1 0 0
337
Exception causes Fault -- Not implemented
EXAMPLE.A
0 0 300 2125 1 0 0
338
Operand Evaluation causes Fault
EXAMPLE.A
0 0 375 2125 1 2 2
342
OVEN_BUF
EXAMPLE.A
0 0 75 2210 1 0 0
343
Relation caused the fault
EXAMPLE.A
0 0 150 2210 1 0 1
345
2
EXAMPLE.A
0 0 0 2295 1 0 0
313
Previous statements caused fault
EXAMPLE.A
0 0 1200 1955 1 1 2
325
Last Statement did not mask fault
EXAMPLE.A
0 0 525 2040 1 0 0
326
Sequence prior to last caused fault
EXAMPLE.A
0 0 600 2040 1 0 0
184
Other ELSIF's caused the fault
EXAMPLE.A
0 0 2325 1445 1 0 0
211
ELSIF caused the fault
EXAMPLE.A
0 0 1950 1530 1 1 2
231

Previous conditions evaluated to false
EXAMPLE.A
0 0 1425 1615 1 0 0
232
Current/future ELSIF caused the fault
EXAMPLE.A
0 0 1500 1615 1 2 3
251
Evaluation of Elsif condition caused fault
EXAMPLE.A
0 0 1350 1700 1 0 0
272
Relation caused the fault
EXAMPLE.A
0 0 1350 1785 1 0 1
295
Relation caused the fault
EXAMPLE.A
0 0 1500 1570 1 2 2
114
OVEN_TEMP
EXAMPLE.A
0 0 1275 1955 1 0 0
315
Addition/Subtraction Fault
EXAMPLE.A
0 0 1350 1955 1 2 3
327
GOAL_TEMP
EXAMPLE.A
0 0 675 2040 1 0 0
328
Subtraction Fault
EXAMPLE.A
0 0 750 2040 1 0 0
329
10
EXAMPLE.A
0 0 825 2040 1 0 0
252
Current ELSIF caused the fault
EXAMPLE.A
0 0 1425 1700 1 1 2
273
Current ELSIF condition caused the fault
EXAMPLE.A
0 0 1425 1785 1 0 0
274
Current ELSIF sequence of statements caused the fault
EXAMPLE.A
0 0 1500 1785 1 0 0

296
Sequence of statements caused fault
EXAMPLE.A
0 0 1575 1870 1 2 2
316
Last statement caused fault
EXAMPLE.A
0 0 1425 1955 1 0 1
330
Raise Statement Caused Fault
EXAMPLE.A
0 0 900 2040 1 2 2
339
Wrong Exception Raised Caused Fault
EXAMPLE.A
0 0 450 2125 1 0 0
340
Exception Handler Caused Fault
EXAMPLE.A
0 0 525 2125 1 0 1
344
OVEN_NOT_RESPONDING
EXAMPLE.A
0 0 225 2210 1 0 0
317
Previous statements caused fault
EXAMPLE.A
0 0 1500 1955 1 1 2
331
Last Statement did not mask fault
EXAMPLE.A
0 0 975 2040 1 0 0
332
Sequence prior to last caused fault
EXAMPLE.A
0 0 1050 2040 1 0 0
253
Other ELSIF's caused the fault
EXAMPLE.A
0 0 1300 1700 1 0 0
59
Previous statements caused fault
EXAMPLE.A
0 0 825 850 1 1 2
69
Last Statement did not mask fault
EXAMPLE.A
0 0 675 935 1 0 0
70
Sequence prior to last caused fault
EXAMPLE.A

0 0 750 935 1 0 0
12
Condition true past n-1
EXAMPLE.A
0 0 225 340 1 1 2
15
Condition true at n-1 iteration
EXAMPLE.A
0 0 150 425 1 0 0
16
Sequence of statements kept condition true
EXAMPLE.A
0 0 225 425 1 0 1
21
Sequence of statements caused fault
EXAMPLE.A
0 0 300 510 1 2 2
27
Last statement caused fault
EXAMPLE.A
0 0 375 595 1 0 1
35
Loop Statement caused fault
EXAMPLE.A
0 0 450 680 1 2 3
44
Loop never executed
EXAMPLE.A
0 0 450 765 1 0 0
45
Loop condition evaluation caused fault
EXAMPLE.A
0 0 525 765 1 2 2
54
Wrong Type of loop used
EXAMPLE.A
0 0 450 850 1 0 0
55
Iteration scheme caused fault
EXAMPLE.A
0 0 525 850 1 2 1
65
Parameter specification caused fault
EXAMPLE.A
0 0 375 935 1 2 2
78
Identifier specified caused fault
EXAMPLE.A
0 0 525 1020 1 0 1
90
1

EXAMPLE.A
0 0 525 1105 1 0 0
79
Discreet range caused fault
EXAMPLE.A
0 0 600 1020 1 0 1
91
Range specified caused fault
EXAMPLE.A
0 0 600 1105 1 2 2
105
1
EXAMPLE.A
0 0 600 1190 1 0 0
106
SAMPLING_DURATION
EXAMPLE.A
0 0 675 1190 1 0 0
46
Nth Iteration caused fault
EXAMPLE.A
0 0 600 765 1 1 2
56
Sequence of statements caused fault
EXAMPLE.A
0 0 600 850 1 0 1
57
Condition true past n-1
EXAMPLE.A
0 0 675 850 1 1 2
66
Condition true at n-1 iteration
EXAMPLE.A
0 0 450 935 1 0 0
67
Sequence of statements kept condition true
EXAMPLE.A
0 0 525 935 1 0 1
28
Previous statements caused fault
EXAMPLE.A
0 0 450 595 1 1 2
36
Last Statement did not mask fault
EXAMPLE.A
0 0 525 680 1 0 0
37
Sequence prior to last caused fault
EXAMPLE.A
0 0 600 680 1 0 1
47

Sequence of statements caused fault
EXAMPLE.A
0 0 675 765 1 2 2
58
Last statement caused fault
EXAMPLE.A
0 0 750 850 1 0 1
68
If statement caused fault
EXAMPLE.A
0 0 600 935 1 2 3
80
Evaluation of condition caused fault
EXAMPLE.A
0 0 675 1020 1 0 0
92
Relation caused the fault
EXAMPLE.A
0 0 675 1105 1 0 1
107
And Relation caused the fault
EXAMPLE.A
0 0 750 1190 1 2 2
121
Left or Prior relation caused fault
EXAMPLE.A
0 0 675 1275 1 0 1
144
Relation caused the fault
EXAMPLE.A
0 0 1125 1360 1 2 2
170
OVEN_TEMP
EXAMPLE.A
0 0 1275 1445 1 0 0
171
Addition/Subtraction Fault
EXAMPLE.A
0 0 1350 1445 1 2 3
195
GOAL_TEMP
EXAMPLE.A
0 0 750 1530 1 0 0
196
Subtraction Fault
EXAMPLE.A
0 0 825 1530 1 0 0
197
10
EXAMPLE.A
0 0 900 1530 1 0 0

122
Right relation caused fault
EXAMPLE.A
0 0 750 1275 1 0 1
145
Relation caused the fault
EXAMPLE.A
0 0 1200 1360 1 2 2
172
LAST_CMD
EXAMPLE.A
0 0 1425 1445 1 0 0
173
1
EXAMPLE.A
0 0 1500 1445 1 0 0
81
Condition true and statements caused fault
EXAMPLE.A
0 0 750 1020 1 1 2
93
If condition true
EXAMPLE.A
0 0 750 1105 1 0 0
94
If statements caused fault
EXAMPLE.A
0 0 825 1105 1 0 0
108
Sequence of statements caused fault
EXAMPLE.A
0 0 825 1190 1 2 2
123
Last statement caused fault
EXAMPLE.A
0 0 825 1275 1 0 1
146
Assignment Statement Fault
EXAMPLE.A
0 0 1275 1360 1 2 3
174
Change in values caused Fault
EXAMPLE.A
0 0 1575 1445 1 0 0
175
Exception causes Fault -- Not implemented
EXAMPLE.A
0 0 1650 1445 1 0 0
176
Operand Evaluation causes Fault
EXAMPLE.A

0 0 1725 1445 1 2 2
198
LAST_CMD
EXAMPLE.A
0 0 975 1530 1 0 0
199
Relation caused the fault
EXAMPLE.A
0 0 1050 1530 1 0 1
223
1
EXAMPLE.A
0 0 825 1615 1 0 0
124
Previous statements caused fault
EXAMPLE.A
0 0 900 1275 1 1 2
147
Last Statement did not mask fault
EXAMPLE.A
0 0 1350 1360 1 0 0
148
Sequence prior to last caused fault
EXAMPLE.A
0 0 1425 1360 1 0 1
177
Sequence of statements caused fault
EXAMPLE.A
0 0 1800 1445 1 2 2
200
Last statement caused fault
EXAMPLE.A
0 0 1125 1530 1 0 1
224
Assignment Statement Fault
EXAMPLE.A
0 0 900 1615 1 2 3
243
Change in values caused Fault
EXAMPLE.A
0 0 750 1700 1 0 0
244
Exception causes Fault -- Not implemented
EXAMPLE.A
0 0 825 1700 1 0 0
245
Operand Evaluation causes Fault
EXAMPLE.A
0 0 900 1700 1 2 2
265
OVEN_BUF

EXAMPLE.A
0 0 825 1785 1 0 0
266
Relation caused the fault
EXAMPLE.A
0 0 900 1785 1 0 1
286
1
EXAMPLE.A
0 0 825 1870 1 0 0
201
Previous statements caused fault
EXAMPLE.A
0 0 1200 1530 1 1 2
225
Last Statement did not mask fault
EXAMPLE.A
0 0 975 1615 1 0 0
226
Sequence prior to last caused fault
EXAMPLE.A
0 0 1050 1615 1 0 0
82
ELSIF caused the fault
EXAMPLE.A
0 0 825 1020 1 1 2
95
Previous conditions evaluated to false
EXAMPLE.A
0 0 900 1105 1 0 0
96
Current/future ELSIF caused the fault
EXAMPLE.A
0 0 975 1105 1 2 3
109
Evaluation of Elsif condition caused fault
EXAMPLE.A
0 0 900 1190 1 0 0
125
Relation caused the fault
EXAMPLE.A
0 0 975 1275 1 0 1
149
Relation caused the fault
EXAMPLE.A
0 0 1500 1360 1 2 2
178
OVEN_TEMP
EXAMPLE.A
0 0 1875 1445 1 0 0
179

Addition/Subtraction Fault
EXAMPLE.A
0 0 1950 1445 1 2 3
202
GOAL_TEMP
EXAMPLE.A
0 0 1275 1530 1 0 0
203
Addition Fault
EXAMPLE.A
0 0 1350 1530 1 0 0
204
10
EXAMPLE.A
0 0 1425 1530 1 0 0
110
Current ELSIF caused the fault
EXAMPLE.A
0 0 975 1190 1 1 2
126
Current ELSIF condition caused the fault
EXAMPLE.A
0 0 1050 1275 1 0 0
127
Current ELSIF sequence of statements caused the fault
EXAMPLE.A
0 0 1125 1275 1 0 0
150
Sequence of statements caused fault
EXAMPLE.A
0 0 1575 1360 1 2 2
180
Last statement caused fault
EXAMPLE.A
0 0 2025 1445 1 0 1
205
Raise Statement Caused Fault
EXAMPLE.A
0 0 1500 1530 1 2 2
227
Wrong Exception Raised Caused Fault
EXAMPLE.A
0 0 1125 1615 1 0 0
228
Exception Handler Caused Fault
EXAMPLE.A
0 0 1200 1615 1 0 1
246
OVEN_NOT_RESPONDING
EXAMPLE.A
0 0 975 1700 1 0 0

181
Previous statements caused fault
EXAMPLE.A
0 0 2100 1445 1 1 2
206
Last Statement did not mask fault
EXAMPLE.A
0 0 1575 1530 1 0 0
207
Sequence prior to last caused fault
EXAMPLE.A
0 0 1650 1530 1 0 0
111
Other ELSIF's caused the fault
EXAMPLE.A
0 0 1050 1190 1 0 0
128
ELSIF caused the fault
EXAMPLE.A
0 0 1200 1275 1 1 2
151
Previous conditions evaluated to false
EXAMPLE.A
0 0 1650 1360 1 0 0
152
Current/future ELSIF caused the fault
EXAMPLE.A
0 0 1725 1360 1 2 3
182
Evaluation of Elsif condition caused fault
EXAMPLE.A
0 0 2175 1445 1 0 0
208
Relation caused the fault
EXAMPLE.A
0 0 1725 1530 1 0 1
229
And Relation caused the fault
EXAMPLE.A
0 0 1275 1615 1 2 2
247
Left or Prior relation caused fault
EXAMPLE.A
0 0 1050 1700 1 0 1
267
Relation caused the fault
EXAMPLE.A
0 0 975 1785 1 2 2
287
OVEN_TEMP
EXAMPLE.A

0 0 900 1870 1 0 0
288
Addition/Subtraction Fault
EXAMPLE.A
0 0 975 1870 1 2 3
307
GOAL_TEMP
EXAMPLE.A
0 0 750 1955 1 0 0
308
Addition Fault
EXAMPLE.A
0 0 825 1955 1 0 0
309
10
EXAMPLE.A
0 0 900 1955 1 0 0
248
Right relation caused fault
EXAMPLE.A
0 0 1125 1700 1 0 1
268
Relation caused the fault
EXAMPLE.A
0 0 1050 1785 1 2 2
289
LAST_CMD
EXAMPLE.A
0 0 1050 1870 1 0 0
290
2
EXAMPLE.A
0 0 1125 1870 1 0 0
183
Current ELSIF caused the fault
EXAMPLE.A
0 0 2250 1445 1 1 2
209
Current ELSIF condition caused the fault
EXAMPLE.A
0 0 1800 1530 1 0 0
210
Current ELSIF sequence of statements caused the fault
EXAMPLE.A
0 0 1875 1530 1 0 0
230
Sequence of statements caused fault
EXAMPLE.A
0 0 1350 1615 1 2 2
249
Last statement caused fault

EXAMPLE.A
0 0 1200 1700 1 0 1
269
Assignment Statement Fault
EXAMPLE.A
0 0 1125 1785 1 2 3
291
Change in values caused Fault
EXAMPLE.A
0 0 1200 1870 1 0 0
292
Exception causes Fault – Not implemented
EXAMPLE.A
0 0 1275 1870 1 0 0
293
Operand Evaluation causes Fault
EXAMPLE.A
0 0 1350 1870 1 2 2
310
LAST_CMD
EXAMPLE.A
0 0 975 1955 1 0 0
311
Relation caused the fault
EXAMPLE.A
0 0 1050 1955 1 0 1
323
2
EXAMPLE.A
0 0 375 2040 1 0 0
250
Previous statements caused fault
EXAMPLE.A
0 0 1275 1700 1 1 2
270
Last Statement did not mask fault
EXAMPLE.A
0 0 1200 1785 1 0 0
271
Sequence prior to last caused fault
EXAMPLE.A
0 0 1275 1785 1 0 1
294
Sequence of statements caused fault
EXAMPLE.A
0 0 1425 1870 1 2 2
312
Last statement caused fault
EXAMPLE.A
0 0 1125 1955 1 0 1
324

Assignment Statement Fault
EXAMPLE.A
0 0 450 2040 1 2 3
336
Change in values caused Fault
EXAMPLE.A
0 0 225 2125 1 0 0
337
Exception causes Fault – Not implemented
EXAMPLE.A
0 0 300 2125 1 0 0
338
Operand Evaluation causes Fault
EXAMPLE.A
0 0 375 2125 1 2 2
342
OVEN_BUF
EXAMPLE.A
0 0 75 2210 1 0 0
343
Relation caused the fault
EXAMPLE.A
0 0 150 2210 1 0 1
345
2
EXAMPLE.A
0 0 0 2295 1 0 0
313
Previous statements caused fault
EXAMPLE.A
0 0 1200 1955 1 1 2
325
Last Statement did not mask fault
EXAMPLE.A
0 0 525 2040 1 0 0
326
Sequence prior to last caused fault
EXAMPLE.A
0 0 600 2040 1 0 0
184
Other ELSIF's caused the fault
EXAMPLE.A
0 0 2325 1445 1 0 0
211
ELSIF caused the fault
EXAMPLE.A
0 0 1950 1530 1 1 2
231
Previous conditions evaluated to false
EXAMPLE.A
0 0 1425 1615 1 0 0

232
Current/future ELSIF caused the fault
EXAMPLE.A
0 0 1500 1615 1 2 3
251
Evaluation of Elsif condition caused fault
EXAMPLE.A
0 0 1350 1700 1 0 0
272
Relation caused the fault
EXAMPLE.A
0 0 1350 1785 1 0 1
295
Relation caused the fault
EXAMPLE.A
0 0 1500 1870 1 2 2
314
OVEN_TEMP
EXAMPLE.A
0 0 1275 1955 1 0 0
315
Addition/Subtraction Fault
EXAMPLE.A
0 0 1350 1955 1 2 3
327
GOAL_TEMP
EXAMPLE.A
0 0 675 2040 1 0 0
328
Subtraction Fault
EXAMPLE.A
0 0 750 2040 1 0 0
329
10
EXAMPLE.A
0 0 825 2040 1 0 0
252
Current ELSIF caused the fault
EXAMPLE.A
0 0 1425 1700 1 1 2
273
Current ELSIF condition caused the fault
EXAMPLE.A
0 0 1425 1785 1 0 0
274
Current ELSIF sequence of statements caused the fault
EXAMPLE.A
0 0 1500 1785 1 0 0
296
Sequence of statements caused fault
EXAMPLE.A

0 0 1575 1870 1 2 2
316
Last statement caused fault
EXAMPLE.A
0 0 1425 1955 1 0 1
330
Raise Statement Caused Fault
EXAMPLE.A
0 0 900 2040 1 2 2
339
Wrong Exception Raised Caused Fault
EXAMPLE.A
0 0 450 2125 1 0 0
340
Exception Handler Caused Fault
EXAMPLE.A
0 0 525 2125 1 0 1
344
OVEN_NOT_RESPONDING
EXAMPLE.A
0 0 225 2210 1 0 0
317
Previous statements caused fault
EXAMPLE.A
0 0 1500 1955 1 1 2
331
Last Statement did not mask fault
EXAMPLE.A
0 0 975 2040 1 0 0
332
Sequence prior to last caused fault
EXAMPLE.A
0 0 1050 2040 1 0 0
253
Other ELSIF's caused the fault
EXAMPLE.A
0 0 1500 1700 1 0 0
59
Previous statements caused fault
EXAMPLE.A
0 0 825 850 1 1 2
69
Last Statement did not mask fault
EXAMPLE.A
0 0 675 935 1 0 0
70
Sequence prior to last caused fault
EXAMPLE.A
0 0 750 935 1 0 0
2
Previous statements caused fault

EXAMPLE.A
0 0 75 85 1 1 2
4
Last Statement did not mask fault
EXAMPLE.A
0 0 75 170 1 0 0
5
Sequence prior to last caused fault
EXAMPLE.A
0 0 150 170 1 0 0

LIST OF REFERENCES

[Bro 88] Brown, Michael L., "Software Systems Safety and Human Errors", *Proceedings of IEEE Compass '88*, pp. 19 - 28, 1988.

[Bry 88] Bryan, William and Siegel, Stan, "Software Product Assurance -- Reducing Software Risk in Critical Systems", *Proceedings of IEEE Compass '88*, pp. 67 - 74, 1988.

[Cha 91] Cha, Stephen S., *A Safety-Critical Software Design and Verification Technique*, Ph. D. Dissertation, Department of Information and Computer Science, University of California, Irvine, Irvine, CA, 1991

[Eri 81] Erickson, C. A., "Software and System Safety", in *Proceedings of the 5th International System Safety Conference, Denver, CO.*, Volume 1, Part 1, System Safety Society, pp III-B-1 to III-B-11, 1981.

[Fri 93] Friedman, Michael A., "Automated Software Fault-Tree Analysis of Pascal Programs", *1993 Proceedings Annual Reliability and Maintainability Symposium*, pp. 458 - 461, 1993.

[Gri 81] Griggs, J. G., "A Method of Software Safety Analysis", in *Proceedings of the 5th International System Safety Conference, Denver, CO.*, Volume 1, Part 1, System Safety Society, pp III-D-1 to III-D-18, 1981.

[Fu' 73] Fussell, J. B., "A Formal Methodology for Fault Tree Construction", *Science and Engineering*, Volume 52, pp. 421 - 432, July 1973.

[Ill 90] Illingworth, Valerie, *Dictionary of Computing*, pp. 250 and 330 - 331, Oxford University Press, 1990.

[Lev 86] Leveson, Nancy G., "Software Safety: Why, what, and How", *ACM Computing Surveys*, Volume 18, Number 2, pp. 125 - 163, June 1986.

[Lev 87] Leveson, Nancy G., Cha, Stephen S., and Shimeall, Timothy J., "Fault Tree Analysis Applied to Ada", Department of Information and Computer Science, University of California, Irvine, Irvine, CA, pp. 1 - 31, November 11, 1987

[Lev 89] Leveson, Nancy G. Cha, Stephen S., and Shimeall, Timothy J., "Safety Verification in Murphy using Fault Tree Analysis", *Tutorial: Software Risk Management*, IEEE Computer Society Press, pp. 394 - 403, 1989.

[Lev 91] Leveson, Nancy G., Cha, Stephen S., and Shimeall, Timothy J., "Safety Verification of Ada Programs Using Software Fault Trees", *IEEE Software*, pp. 48 - 59, July 1991.

[Lev 83] Leveson, Nancy G. and Harvey, P. R., "Analyzing Software Safety", *IEEE Transactions on Software Engineering*, Volume SE-9, Number 5, pp. 569 - 579, September 1983.

[Lub 88] Lubbes, H. O., et al., "Compass Past and Future", *Proceedings of IEEE Compass '88*, pp. V - VII, 1988.

[Mcc 88] McCarthy, Roger L., "Present and Future Safety Challenges of Computer Control", *Proceedings of IEEE Compass'88*, pp. 1 - 7, 1988.

[Mcg 89] McGraw, Richard J., *Petri Net and Fault Tree Analysis: Combining two Techniques for a Software Safety Analysis on an Embedded Military Application*, Master's Thesis, Naval Postgraduate School, Monterey, CA, December, 1989.

[Mci 83] United States Air Force, Technical Report, "Fault Tree Techniques as Applied to Software (Soft Tree)", McIntee, J. W. Jr., pp. 1 - 12, March 1983.

[Nee 90] Needham, Donald M., *A Formal Approach to Hazard Decomposition in Software Fault Tree Analysis*, Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1990.

[Ngu 88] Nguyen, Thieu and Forester, Kari, *Alex -- An Ada Lexical Analysis Generator, Version 1.0*, Arcadia Environment Research Project, Department of Information and Computer Science, University of California, Irvine, Irvine, CA, pp. 1 - 12, May 1988.

[Par 90] Parnas, D. L., Van Shouwen, J. A., and Kwan, S. P., "Evaluation of Safety-Critical Software", *Communications of the ACM*, Volume 33, Number 6, pp. 636 - 648, June 1990.

[Rol 86] Department of Information and Computer Science, University of California, Irvine, Irvine, CA, Technical Report Number 86-06, "Software Fault Tree Analysis Tool User's Manual", Rolandelli, Craig, Shimeall, Timothy J., Genung, Christi, and Leveson, Nancy, pp. 1 - 6, 1986.

[Sal 76] Salem, S. L., *A Computer Oriented Approach to Fault Tree Construction*, Ph. D. Dissertation, University of California, Los Angeles, Los Angeles, CA, 1976.

[Tab 88] Tabeck, David and Tolani, Deepak, *Ayacc User's Manual Version 1.0*, Arcadia Environment Research Project, Department of Information and Computer Science, University of California, Irvine, Irvine, CA, pp. 1 - 15, May, 1988.

[Tay 82] Taylor, J. R., *Fault Tree and Cause Consequence Analysis for Control Software Validation*, Riso National Laboratory, DK-4000 Doskilde, Denmark, pp.5 - 17, January 1982.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
Dudley Knox Library Code 052 Naval Postgraduate School Monterey, CA 93943-5002	2
Dr. Timothy J. Shimeall Computer Science Department Code CS/SM Naval Postgraduate School Monterey, CA 93943	4
Dr. David A. Erickson Computer Science Department Code CS/ER Naval Postgraduate School Monterey, CA 93943	2
Mr. Bob F. Westbrook (Code 31) Naval Air Warfare Center, Weapons Division China Lake, California 93555	1
Ms. Eileen T. Takach Naval Air Warfare Center, Aircraft Division, Indpls. 6000 E. 21st Street Indianapolis, Indiana 46219-2189	1
Dr. Michael Friedman 18950 Mt. Castile Circle Fountain Valley, California 92708	1
Dr. Ted Lewis Computer Science Department Code CS/LT Naval Postgraduate School Monterey, CA 93943	1
CPT Robert R. Ordonio 9828 Bristol Avenue Silver Spring, Maryland 20901	1